# Mobile Display of Knitting Patterns

Bachelor's Thesis

Bianca Ploch
540609

16 August 2016

Supervisor:
Prof. Dr. Debora Weber-Wulff

HTW Berlin
International Media and Computing (Bachelor)

# Contents

# Chapter 1

# Introduction

Since the beginning of the 2000s, knitting has encountered a steady rise in popularity, claims an article by Lewis 2011. This, she says, might be due to the rise of the internet and social media, and the increasingly important role they play in the daily life. The older knitting generation is adapting to new technology and switching over, Nielsen explains, bringing knitting as a craft and hobby closer to the younger generations (Lewis 2011). Online communities like Ravelry[1] and Youtube[2] teach the knitting enthusiasts knittings techniques and patterns of all kinds — never has knitting knowledge been more accessible.

Considering this, it is all the more surprising that there are only few apps related to knitting to be found on the Play Store, Google's digital distribution service for Android apps. Only one app supports the creation of a knitting pattern chart. Mobile devices have the potential to be a great help to knitters. An app could help knitters keep track of the projects they are currently knitting, look up instructions, and store knitting patterns. The latter especially aids the mobile knitter — no longer is it necessary to carry sheets of paper with pattern charts or even books, as seen in **Figure 1.1**, around.

[1] `http://www.ravelry.com/`
[2] `https://www.youtube.com/`

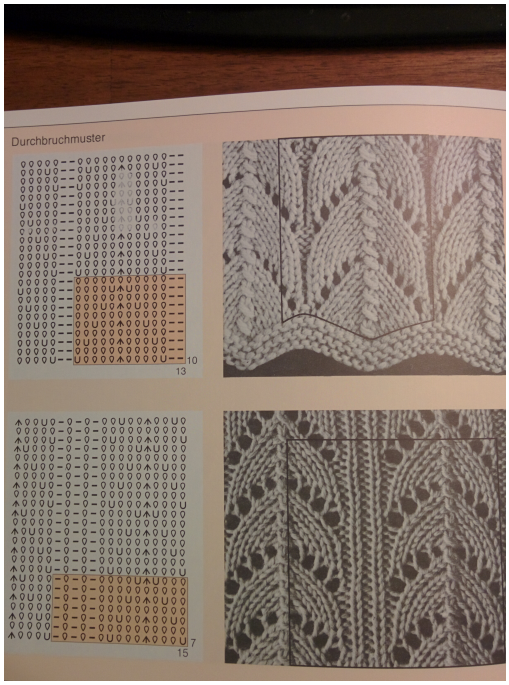



Figure 1.1: Knitting patterns and their corresponding pattern charts from Natter 1983, p142

The difficulty with displaying such a chart on a mobile device is due to the size of the device's screen. This screen is a far smaller medium than a sheet of paper, on which pattern charts are normally printed.

Pattern charts, excluding the smaller charts, are therefore too big to be viewed easily inside an app. The goal for this thesis is to research how a knitting pattern chart can be input and displayed on mobile devices running the Android operating system and develop a working prototype showcasing the results of that research. The prototype will be evaluated through testing by users representative of the prototype's target group. This testing will, if time permits, executed iteratively throughout developement to ensure a useful User Interface (UI) design. The term useful in this context is taken from the article *Usability 101: Introduction to Usability* by Nielsen 2014 — he uses the term to summarize the usability and utility of a design. Nielsen furthermore defines utililty and usability as indicators of "[...] whether [a design] provides the features you need" and "how easy [and] pleasant these features are to use" (Nielsen 2014).

# Chapter 2

# Background

## 2.1 Definition of Knitting Pattern and Knitting Pattern Chart

A knitting pattern specifies a set of instructions outlining the steps necessary to create a knitted textile or fabric. For knitting a fabric the knitter uses two or more knitting needles and a long, continuous strand of yarn which they use to form intersecting loops with, which in turn creates a textile or fabric. "Knitting is a conversion system in which yarn loops are interwoven to form a fabric" (Raz 1993, p17). The type of loops the knitter uses as well as the kind of yarn determine the attributes of the knitted piece: elasticity, form and texture. A knitted fabric can be stretched in both horizontal and vertical directions, as well as the directions in-between. This makes it stand apart from woven fabric, which is created by layering two threads in an interlaced manner. The woven cloth is very limited in its ability to stretch and be formed, excluding, of course, the usage of stretching threads.

Knitting patterns can come in form of written instructions, usually with abbreviations used for the stitch terms, e.g. k2tog for the "knit two together" stitch, or in form of a pattern chart which consists of a grid filled with symbols. Both written patterns and pattern charts, are generally split into rows, where each row has a definite number of stitches. Each cell in such a grid signifies a stitch in the pattern and the symbol displayed in a cell corresponds with the stitch that needs to be made in that place in the pattern. In what order the rows have to be knitted depends on the chart type; some charts display only the uneven numbered rows, which belong to the right side (RS) of

the knitted fabric, and expect the knitter to knit the return row on the wrong side (WS) inverse to the RS, i.e. knits would be knitted as purls and purls as knits. Other charts show all rows, the uneven numbered for the RS and the even numbered ones for the WS.

So far there does not exist an international standard for the symbols used in knitting charts or the abbreviations in written instructions. Symbols used by the industry usually vary depending on the region (Raz 1993, p57) and it is the norm that a knitting pattern includes a glossary for the symbols and abbreviations used in the pattern. One exception to this is Japan, where there exists a Japanese Industrial Standard on knitting symbols used in the industry and for the hobby hand knitters: JIS L 0201-1995 (Association 1995). This leads to the fact that Japanese knitting pattern charts are published without an index of the symbols used.

Other regional industry standards that Raz mentions in his book are the German Standard and the needle notation system, "the most explicit and accurate of all notation systems" (Raz 1993, p58), which is solely used for industrial knitting machines and shows the positions of the needles of the knitting machine for each stitch.

## 2.2 Comparison of Existing Solutions

### 2.2.1 Android apps

When searching for the term "knitting" in the Google Play Store, Android's official source for Google-approved applications, few results pop up. Next to a surprising amount of games about knitting, there are apps for knitting counters, knitting patterns and knitting instructions for those who wish to begin knitting. The following sections will look at the top five apps for creating and managing knitting projects with row counters and pattern display, as well as knitting chart creation.

#### knit tink — Row Counter by Jennifer K. Warren

*The app can be found at* `https://play.google.com/store/apps/details?id=com.warrencollective.knittink` *(last accessed: 2016-08-11)*

Out of all most popular knitting apps, knit tink features the most modern and clean design. The app can be used for free or bought as an ad-free pro version. Features include the creation, editing, viewing, and deletion of projects, the setup of one row,

and one repeat counter per project, as well as the unlinking of the row counter from the repeat counter. The free version of the app restricts the number of projects to three. The developer announced an on-screen display of a knitting chart in PDF format as an upcoming feature.


(a) Row counter

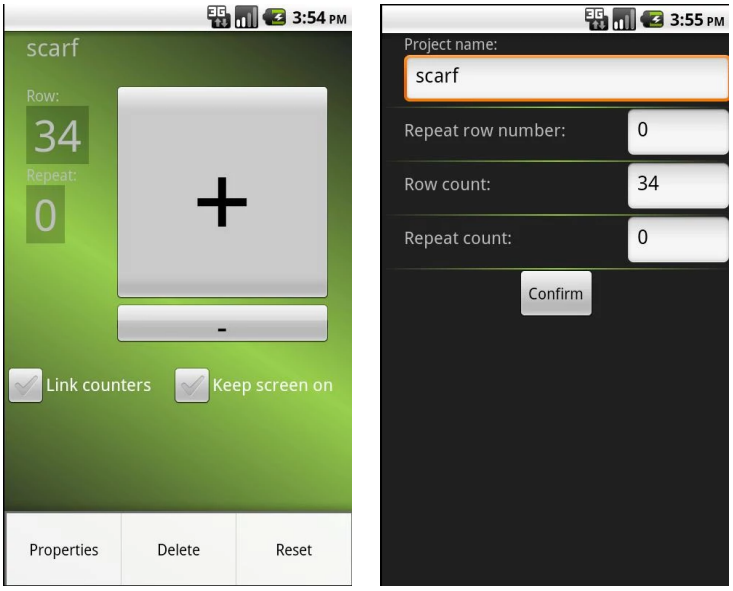
(b) Project setup from Warren 2015

Figure 2.1: Screenshots of the app knit tink

### Knitting Counter by mkacki

*The app can be found at* `https://play.google.com/store/apps/details?id=org.kuklake.rowCounter` *(last accessed: 2016-08-11)*

Knitting Counter offers the same features as the knit tink app, the only differences being the layout of the user interface and the option to keep the phone from going into sleep mode, i.e., turning the phone screen off.




(a) Row counter (b) Row counter setup

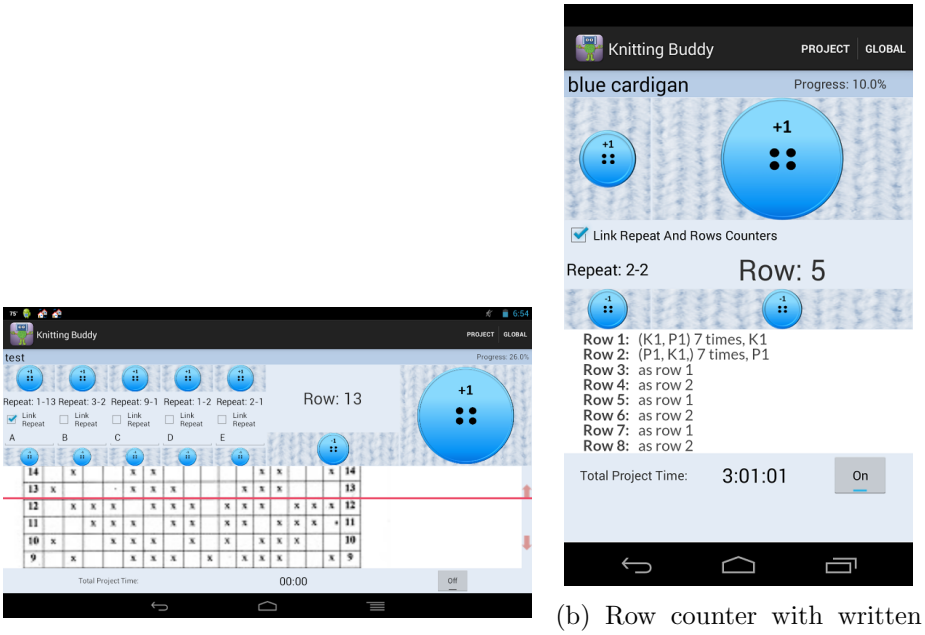
Figure 2.2: Screenshots of the app Knitting Counter

### Knitting and Crochet Buddy by Colorwork Apps

*The app can be found at* `https://play.google.com/store/apps/details?id=androiddeveloperjoe.knittingbuddy` *(last accessed: 2016-08-11)*

The Knitting and Crochet Buddy contains a plethora of features related to knitting and crocheting. As is the standard with the previously mentioned apps, it offers the possibility to manage different knitting and crocheting projects, with each a row and a repeat counter per project. Users can also enter written instructions or add a picture of the pattern chart to be displayed on the counter screen.

Additional features include, but are not limited to: yarn and crochet charts, an abbreviation chart, size charts for knitting needles and crocheting hooks, a project timer, a ruler function, and a flashlight.




(a) Row counter with pattern chart picture

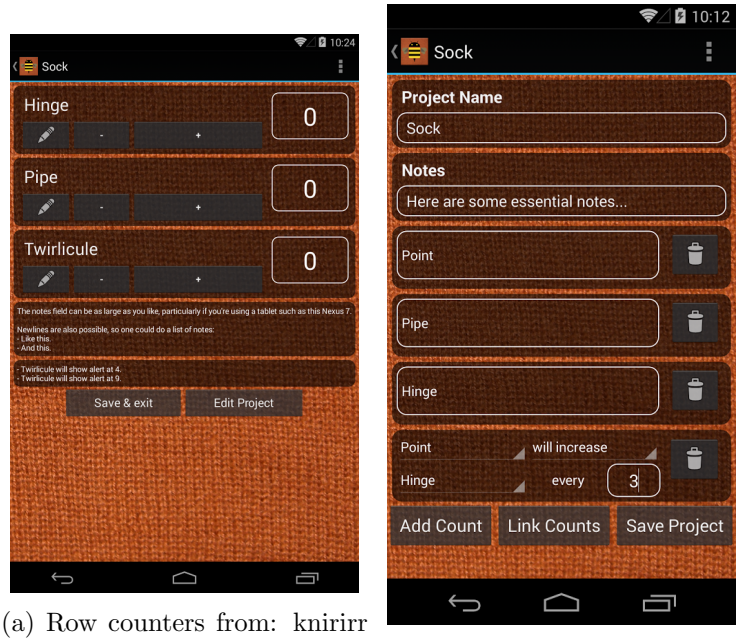
(b) Row counter with written pattern instructions

Figure 2.3: Screenshots of the app Knitting and Crochet Buddy

### BeeCount knitting Counter by knirirr

*The app can be found at* `https://play.google.com/store/apps/details?id=com.knirirr.beecount` *(last accessed: 2016-08-11)*

BeeCount differs from the standard of one row counter per project in that it allows multiple counters. These counters can be for parts of the knit piece that belong to the same project, as is the case, e.g. a knitted sweater. These counters within a project can be linked together, so that increases or decreases in one counter affect the row number of another counter (see **Figure 2.4b**). Furthermore, alerts can be set on counters to be triggered once the counter reaches a set number.




(a) Row counters from: knirirr 2016

(b) Row counters setup

Figure 2.4: Screenshots of the app BeeCount Knitting Counter

### Knitting Chart Maker by Awesome Applications

*The app can be found at* `https://play.google.com/store/apps/details?id=knitting.chart.maker` *(last accessed: 2016-08-11)*

When it comes to pattern charts, none of the aforementioned apps offer a solution to input a knitting pattern chart. Only one app, the *Knitting and Crochet Buddy*, has the option to include a picture of a pattern. Therefore, I looked at Knitting Chart Maker, an app that focuses solely on the creation and editing of charts.

The app has over 30 stitch symbols that the user can use to create a pattern chart. The symbols are defined by the app and are not taken from a standard. The user cannot devise their own stitch symbols. Symbols can be used by selecting the symbol in the left-hand menu and then transferred onto the grid by tapping on a cell. Alternatively, the user can select the paintbrush button on the top-left menu and use their finger to paint the symbols onto every cell touched in a swiping motion, not unlike drawing with a pencil. The whole grid is zoomable up to a certain zoom level.

While in-app, the user can purchase the pro version which allows them to save and export patterns. Charts can be exported in the form of written instructions or a picture. Included are also various sharing features, such as uploading the saved chart to Dropbox, or sharing a chart with a friend, who can then open that chart in their paid copy of the app.

The app is locked in landscape mode and the chart dimensions are limited to 50 x 50. The pattern chart grid is implemented using OpenGl's canvas and drawing images at the cell positions.

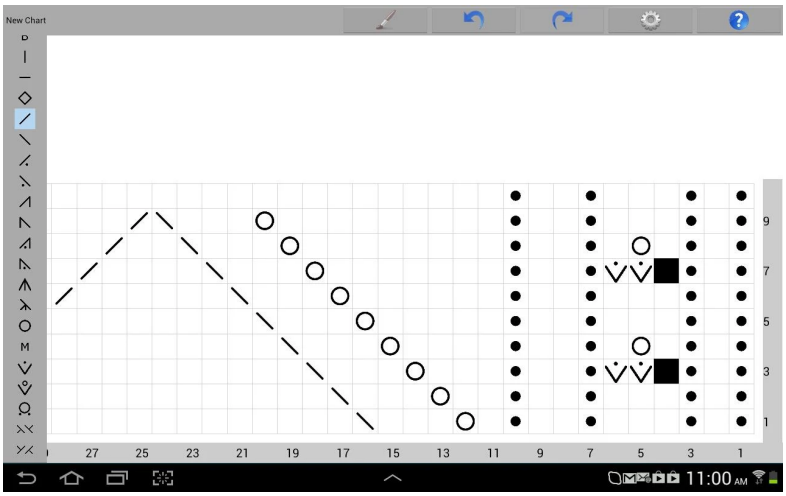



Figure 2.5: Chart editor of Knitting Chart Maker

### 2.2.2 Other

**KnitML by Jonathan Whitall**

*The project can be found at* `http://www.knitml.com/blog/` *(last accessed: 2016-08-11)*

KnitML has no connection to Android, but has an honorable mentioned here since it addresses the problem of inputting a knitting pattern. KnitML is an XML based format for describing the knitting process from beginning to the finished product. With KnitML the project aims to establish an international standard for knitting pattern expression[1]. He aims to do so by using the Knitting Expression Langauge, KEL, that he defined

[1] `http://www.knitml.com/blog/static.php?page=about-knitml`

(Whitall 2009). KEL is based on the Groovy programming language[2] for the Java platform and the GroovyMarkup architecture.

The following KEL expression

```
Pattern {
    generalInformation
}
```

Listing 2.1: Example expression in KnitML

would result in

```
<pattern>
    <general-information/>
</pattern>
```

Listing 2.2: Example expression in KnitM: XML result

The project has not seen updates in any form since 2013 and I presume it discontinued. A beta of an editor program for KEL and its resulting XML can be found on the homepage of the project, knitml.com.

[2] http://groovy-lang.org/templating.html

# Chapter 3

# Requirements

## 3.1 Functional Requirements

The requirements for this thesis are formed from interviews conducted with volunteers at the beginning of this thesis. Three participants, stemming from both the author's acquaintances as well as from volunteers recruited from a poster posted publicly nearby the HTW's campuses, have been interviewed. Prerequisite for a participant in such an interview was a proficiency and an interest in knitting. During a time frame of 45 to 60 minutes the participants were asked to answer a set of questions concerning their knitting experience as well as what features they would like to see in an app aimed to aid them during the creation and viewing of a knitting pattern chart. The catalogue of the questions asked and the answers given by the participants during these interviews can be found in the appendix. From these interviews user stories were formulated and corresponding functional requirements were extracted — see table 3.1 below.

ref here

| # | User Story | Functional Requirement |
|---|---|---|
| 1 | As a knitter I want to be able to see the knitting pattern chart on my phone while knitting | Display of knitting pattern chart that is usable while knitting |
| 2 | As a knitter I want to create my own charts in the app both in a grid format and a row format | Create patterns that support row and grid format |

| # | User Story | Functional Requirement |
|---|---|---|
| 3 | As a knitter I want to transcribe charts from paper into the app with both grid and row formats | Pattern editor |
| 4 | As a knitter I want to have a list of all the patterns in the app and add and remove patterns from that list | CRUD for patterns and showing list of patterns |
| 5 | As a knitter I want to convert metric units for needle sizes, yarn weight and length to imperial and vice versa Unit converter in app | Unit converter in app |
| 6 | As a knitter I would like to enter a set of written knitting instructions and be able to see each individual instruction while knitting and jump to the next instruction with a button press | Editor for written instructions and view of them to be used while knitting with button or voice command |
| 7 | As a knitter I want to use my phone to count the rows I knit | Row counter |
| 8 | As a knitter I would like to be able to look up the explanations and visual instructions for different kinds of stitches while inside the app | Glossary of stitches with explanations and instructions |
| 9 | As a knitter I want to have a way to jump to the row I'm currently on in my knitting pattern and to get back to the default zoom level | Button for resetting the zoom level and to jump to current row when viewing a pattern |
| 10 | As a knitter I want to be able to take pictures of the finished, knitted products of a pattern | In-app camera and function for adding images from disk |
| 11 | As a knitter I want to be able to see pictures of the knitted products of a pattern | Gallery for knitted products from a pattern |

| # | User Story | Functional Requirement |
|---|---|---|
| 12 | As a knitter I want to have all my knitting projects with their details (pattern, required needle size and yarn, etc.) easily accessible in one app | Knitting project management functions |
| 13 | As a knitter I want to be able to use the row counter with another app in the foreground | Have row counter increase and decrease button in notification bar when knitting app is not the active app |
| 14 | As a knitter I want my screen to stay on until I exit the app | Force screen to stay on while in-app |

Within the context of this thesis the focus lies on the functional requirements #1, 2, 3, 4, 7, and 8. The protoype of the app will present a functioning editor as well as a viewer for knitting pattern charts. Two input styles will be available for both viewer and editor: a grid style and a row style. The in-app generated pattern will be stored on disk and will be accessible with CRUD operations within the app. The viewer will have a row counter next to the displayed pattern chart. Buttons for switching between the view styles will be present in the editor as well as the viewer. The option to import and export pattern files will be available as well in case the user wants to move their patterns to or from a different Android device.

After these requirements have been fulfilled and if time allows, additional features for the app will be: a button for resetting the zoom level and jumping back to current line in the pattern, a row counter increase and decrease button outside of the app, and the option to force the screen to stay awake while within the app.

## 3.2 Non-functional Requirements

The prototype will store the pattern files locally on the device the app runs on. All prototype operation will run locally, connectivity to the internet is not needed. Internet connectivity is an option for a later version of the prototype, e.g. for backing the pattern files up to cloud storage and sharing patterns. Since this thesis focuses on the UI part of an app, storage will be restricted to simple, local solutions. This is also done to better fit the time restraints placed in this thesis.

# Chapter 4

# Design

The desired outcome of this thesis will be a working Android app prototype with CRUD functions for knitting chart patterns and a row counter functionality while viewing a pattern. This prototype is intended as an aid for knitters of all backgrounds during their respective knitting projects. Patterns will be saved locally as a JavaScript Object Notation (JSON) file on the device's internal storage. It would also be an option to store patterns on a server and let the app play the role of client, but that would not fit within the time constraints of this thesis. To give the user the ability to backup their patterns the app will support the import and export of pattern from external storage. For a detailed explanation of Android's concepts of internal and external storage see Section 6.3. Patterns exported will be accessible by the user and can be handled in whatever way the user sees fit to, for example, share or upload a pattern.

A chart pattern will consist of a set number of rows and columns. Each cell of the grid contains a symbol representing a knitting stitch. Created pattern are stored locally on the device and can be manipulated by the user in-app, as CRUD operations apply. Creating, editing and viewing patterns will be based on two shared visual formats for the pattern: a grid format and a row format.

The grid format will display the pattern in a grid, simulating the most common form of commercial distribution for knitting chart patterns on both analogue and digital media. Manipulation of the pattern content will be possible through a software keyboard containing the stitch symbols. A symbol can be selected and then applied to cells in the grid via touch. The symbol will stay active until the user selects a different symbol. The grid size can be changed with a button which opens a dialog where the desired amount

of rows and columns can be entered. On confirmation the grid will shrink or expand to the set dimensions. Any symbols lying outside of the new bounds will be deleted, whereas new cells will be empty. The grid will be zoomable to a pre-defined minimum and maximum scale as well as scroll horizontally and vertically.

Similar to the grid format the row format will display the pattern rows, but will forego the representation of the columns. Instead the cells of a row will be summarized in such a way, that consecutive, identical symbols will be represented by a number value equal to the count of the symbols and followed by the stitch symbol. Rows in this format can be edited likein a conventional text editor - a movable cursor to show where further user input will be inserted and text selection functions for multiple character deletion, copying and pasting will be available. The software keyboard corresponding to this format will consist of the stitch symbols and a num pad, as well as an enter and a backspace key. The pattern will support two-dimensional scroll.

Viewing a pattern will come with a row counter below the actual chart pattern. This counter can be increased, decreased and reset by utilizing buttons. The counter is limited between one, as the first row of a pattern, and the number of rows the pattern contains in total. The current row will be indicated through a highlight on the pattern, marking the corresponding row in both grid and row format. When exiting the viewer the current row number will be saved in the pattern file and applied to the counter the next time the pattern is viewed

While editing or viewing a pattern, the row and the grid format will allow the user to 2D scroll, meaning both vertical and horizontal scroll. Additionally, the grid view can be zoomed and reset to default zoom and scroll. Switching between both formats while editing and viewing a pattern will be supported with a button. Upon switching the pattern will be saved. Renaming, deleting, saving, and exporting the pattern will be possible from within both formats with menu entries.

On app launch the list of patterns saved on the device will be shown. Menu entries for exporting all patterns and importing a single pattern will be available on the list screen. For the import the user can choose a file on the device from a file chooser. Exporting will export files to a set directory on the publicly accessible storage of the device. A list item will consist of the pattern name, an edit, and a delete button. A click on the pattern name will open the pattern in the viewer in row format with the default dimensions of 10 columns and 10 rows. Below the list will be a button to create a new

pattern which will open a dialog for entering the new pattern's name. After confirming a name, the editor will open with the row format.

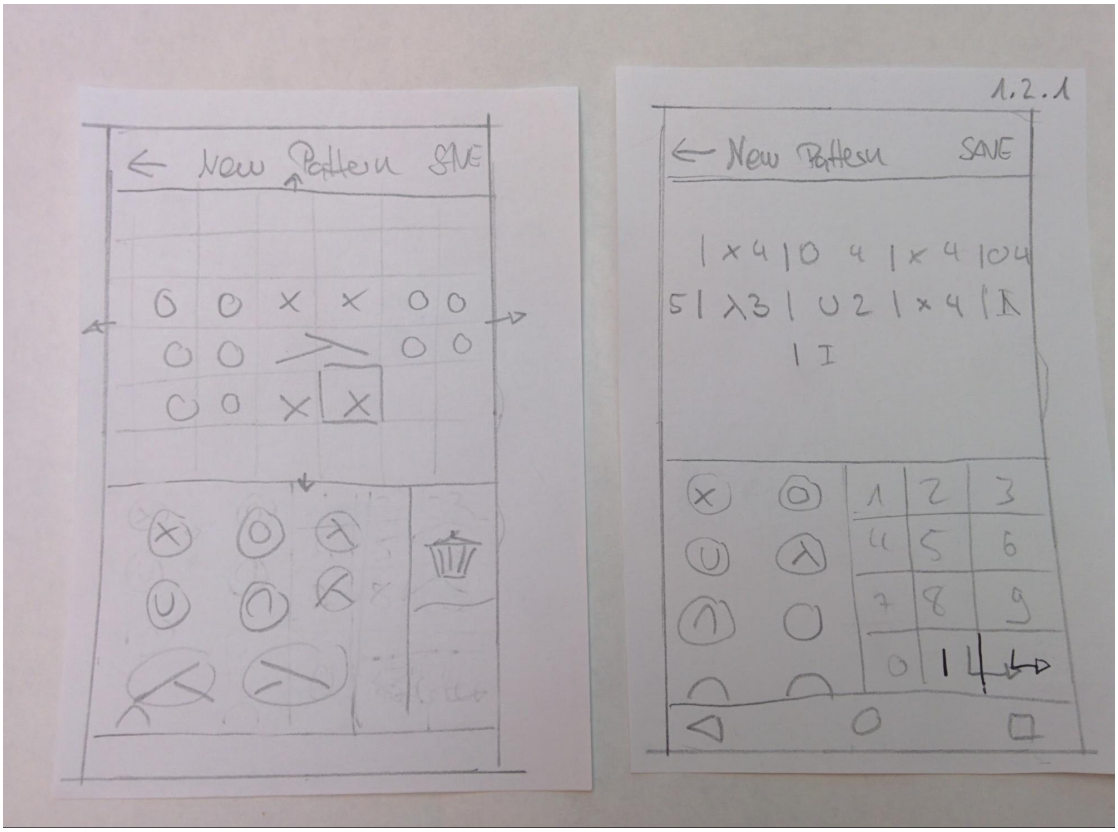



Figure 4.1: Editor screens for grid and row format with pattern name and save button in navigation bar

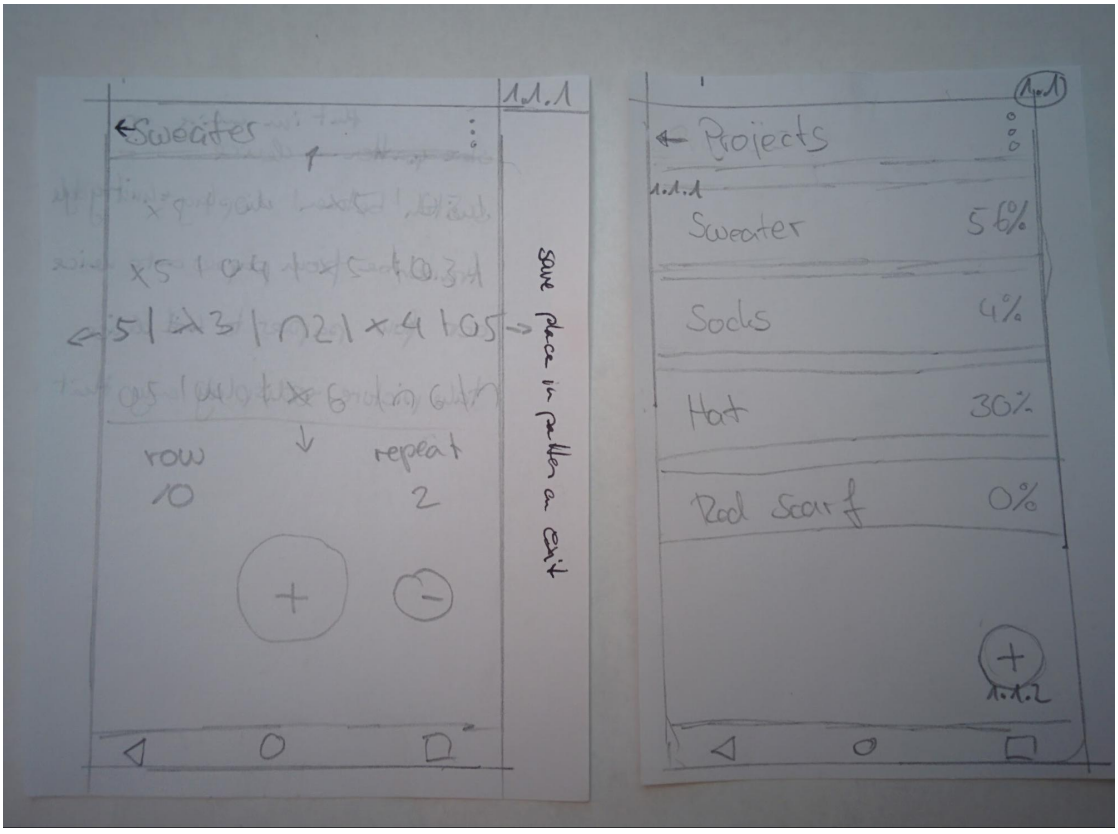



Figure 4.2: Viewer screen for row format and selection screen for stored patterns

Figure 4.3: Screens for editor and viewer without Android elements

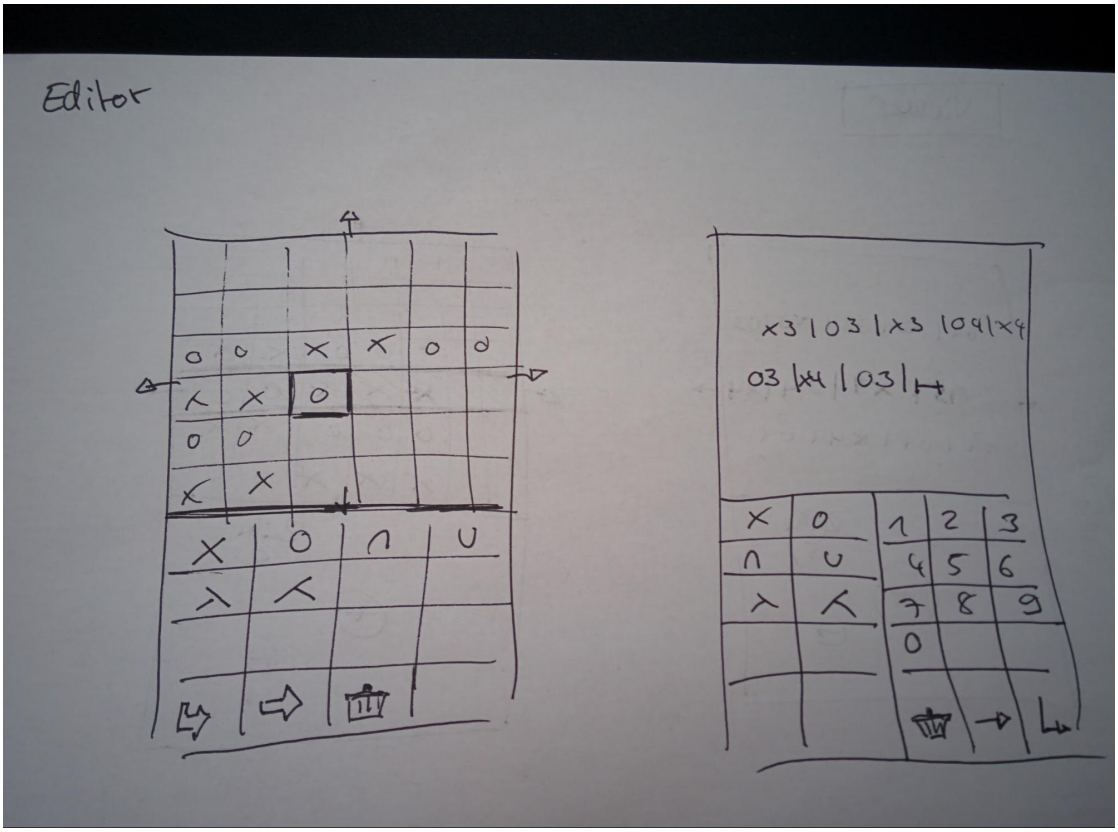



Figure 4.4: Editor screens for grid and row format

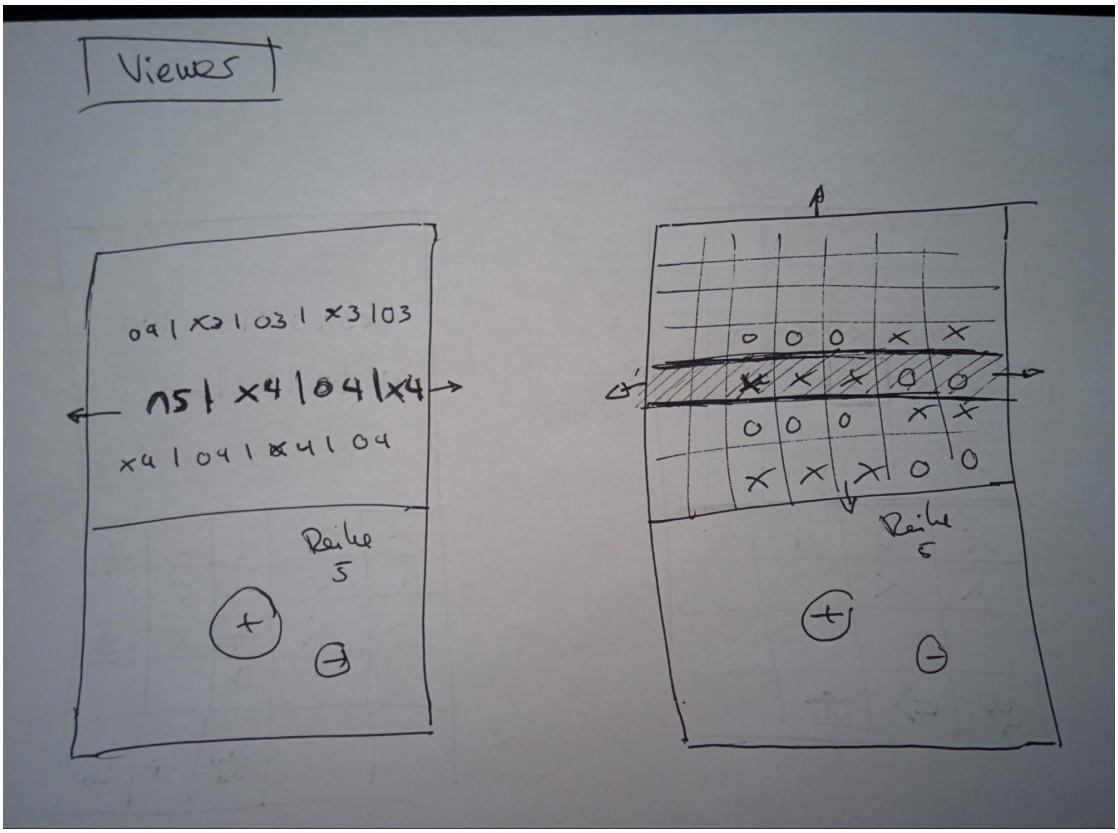



Figure 4.5: Viewer screens for grid and editor format with row counter

# Chapter 5

# Android Basics

## 5.1 The Operating System Android

Android is an open source operating system for mobile phones and tables based on the Linux kernel (Android Developers 2016d). Android apps are distributed on Google Play, a service owned by Google. To build an Android app Google offers the Android Software Development Kit (SDK), containing sample projects, necessary Android libraries and an Android emulator. Additionally, Google recommends to use the official Android Integrated Development Environment (IDE) Android Studio, which is based on IntelliJ IDEA and offers many useful tools, including testing frameworks, a Graphical User Interface (GUI) for screen layouts, and the build tool Gradle (Android Developers 2016j). The concepts and Android components discussed throughout this chapter are taken from the Android Developer reference[1], training[2], and Application Programming Interface (API) guides[3] found online.

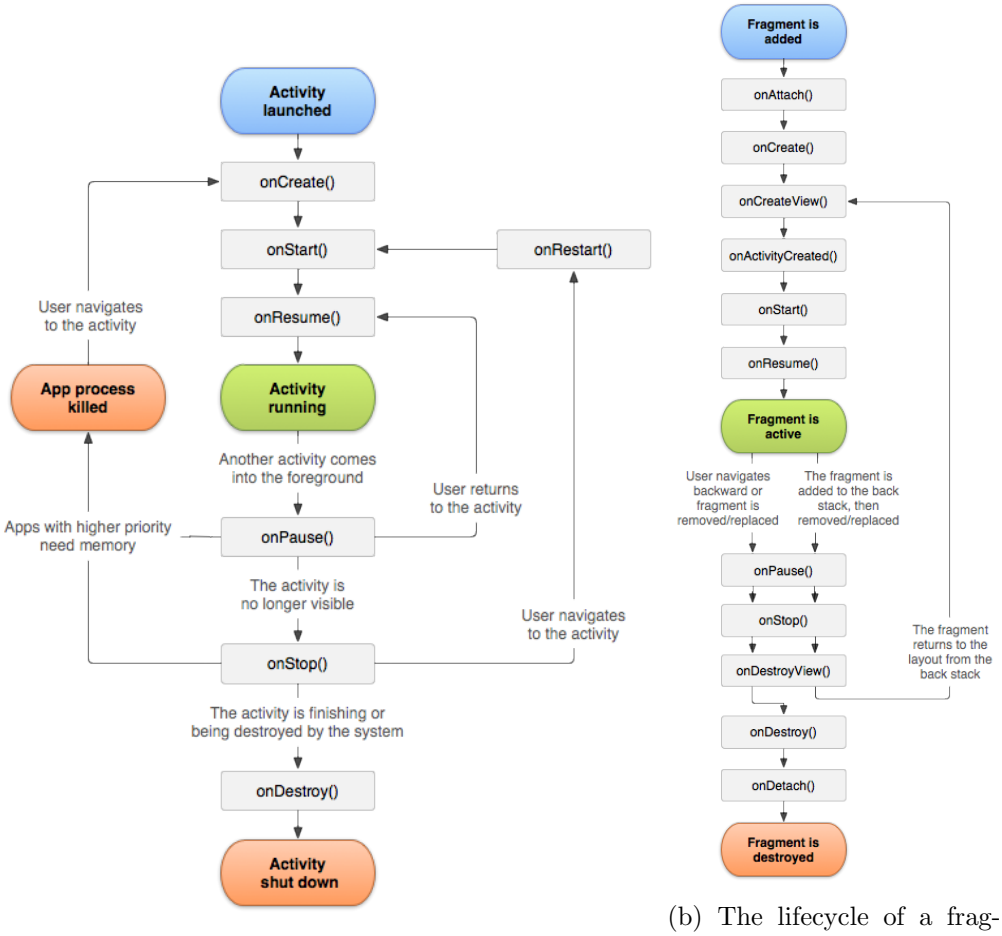
## 5.2 Basic Components of an Android App

### 5.2.1 Activity

The `Activity` class is needed to display any user interface and as such usually has a single purpose — handling a login would be such a purpose. An app consists of one or more activities that are in some way connected to each other (Android Developers

[1] `https://developer.android.com/reference/packages.html` (last accessed 2016-08-11)
[2] `https://developer.android.com/training/index.html` (last accessed 2016-08-11)
[3] `https://developer.android.com/guide/index.html` (last accessed 2016-08-11)

2016b). `ViewGroups` and `Views` can be added to the view hierarchy of activities and fragment — these views define different UI components for Android. An activity can also embed multiple fragments which then live in a viewgroup inside the activity's own view hierarchy (Android Developers 2016h). When containing fragments, the activity's job is that of managing those fragments through getters and setters and orchestrating the communication between fragments, which is done with callbacks defined in the fragments. An activity features methods such as `onCreate()`, `onStart()`, `onStop()`, and `onFinish()`, which can be overwritten to implement logic that is executed at different points in the activity's lifecycle (see **Figure 5.1a**). The same applies for a fragment, but where an activity can stand alone, a fragment always needs to be attached to an activity; it is connected to that activity's lifecycle.




(a) The lifecycle of an activity

(b) The lifecycle of a fragment

Figure 5.1: The lifecycles of activities and fragments

### 5.2.2 Actionbar

The actionbar located at the top of an app and has several important functions. It displays the application name or the title of an activity, houses the action buttons, and the action overflow. Action buttons should contain the most commonly and important actions used in an app (Android Developers 2016a). The action overflow contains action buttons that are hidden from plain view, either because the actionbar was not wide enough to show all buttons or because of a deliberate design decision. Such a decision is usually made when the button in question is connected to an action that is rarely used, e.g. renaming something, or when the action has far-reaching consequences and shouldn't be near buttons that are used frequently, lest the user accidently hits it. Such an action could be the deletion of the pattern currently being edited, such as in the case of a knitting app.

### 5.2.3 Fragment

The `Fragment` class usually implements a specific user interface or behaviour and should, ideally, be modular, so that they can be reused within multiple activities or in different screen configurations. Just like an activity a fragment has its own lifecycle, see **Figure 5.1b**.

A fragment's creation is always embedded in an activity (Android Developers 2016h) — forcing the fragment to pause or stop alongside its parent activity's lifecycle. Fragments can also house viewgroups and views and are intented to function as interchangeable modules, e.g. as UI modules for an app that runs on devices of varying sizes and that wants to present the user with a dynamic UI fit to suit the screen size (see **Figure 5.2**). Android offers different fragment subclassse with predefined behavior, such as the DialogFragment class, that opens a fragment as a floating dialog by default (see **Figure 5.3**).

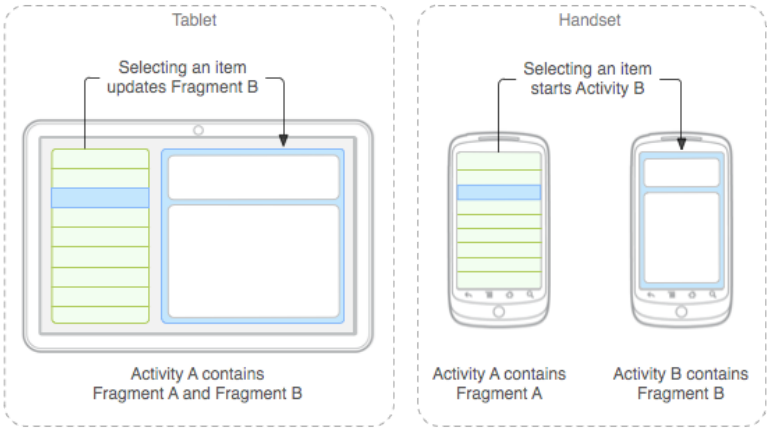



Figure 5.2: Two fragments of one activity and their layout on two different screen sizes

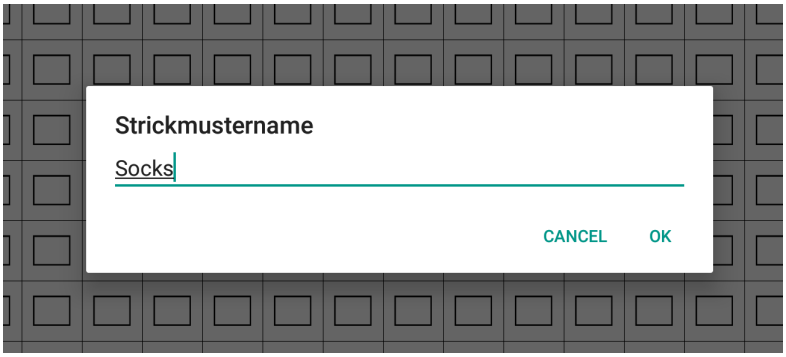



Figure 5.3: A dialog fragment for naming a pattern

Communication between different fragments needs to be handled by the activity, which manages the fragments. An activity communicates with a fragment by keeping a reference to the fragment and calling its public methods. On the other hand the fragment should not possess a reference to its parent activity — instead the parent activity should implement a callback interface defined inside the fragment (Android Developers 2016g). A good practive to enforce the implementation of a fragments callback interface is to check for its existence when the fragment is attached to the activity — example code proposed by the Android Developer guide concerning how to check for this can be seen in *Listing 5.1* (Android Developers 2016g).

```
1 public static class FragmentA extends ListFragment {
```

```
2       OnArticleSelectedListener mListener;
3       ...
4       @Override
5       public void onAttach(Activity activity) {
6           super.onAttach(activity);
7           try {
8               mListener = (OnArticleSelectedListener) activity;
9           } catch (ClassCastException e) {
10              throw new ClassCastException(activity.toString() + " must
                    implement OnArticleSelectedListener");
11          }
12      }
13      ...
14  }
```

Listing 5.1: Example code for enforcing the implementation of a callback interface

### 5.2.4 View

Views are the most basic block that the UI is built from (Android Developers 2016o). A view's bounds are always rectangular and its position is defined by its top and left coordinates with the point of origin at the top left. It is the view's job to handle its drawing and event handling. For this the view has the predefined methods `onDraw()` and `onTouchEvent()`, respectively. The `View` class is also the base for viewgroups which in turn are the base for layouts, containers for other views or viewgroups. The views from a window are arranged in a tree structure. Views can be added to this tree statically, by specifying them in a Extensible Markup Language (XML) layout file, or dynamically, from code. Android comes with plenty of view subclasses, specialized in acting as controls or displaying specific types of content, e.g. text or images. If the pre-existing views don't match a developer's needs, they can also implement a custom view to take control of the drawing and the event handling as it fits their requirements. For this the `onDraw()` method can be overridden and custom operations can then be executed on the `Canvas` object that is contained in the method parameters.

Android ships with many subclasses of View, e.g. the `TextView` class which diplays text content. The view class is also the basis for viewgroups, to which the layouts, e.g. `LinearLayout` and `RelativeLayout`, belong to. Views in a window are bundled together

as a tree with a layout being the top-most root. To add views to an activity or fragment the views can be declared in the corresponding XML layout or from code.

### 5.2.5 Storage

File storage in Android devices is separated into "internal" and "external" storage — this refers to the fact, that Android devices often times have a built-in, non-removable memory and an external, removable medium in the form of an SD or a micro SD card (Android Developers 2016f). This storage separation even exists on devices with only built-in memory — in such cases the storage is partitioned into "internal" and "external" partitions. This assures that the concept of two storages persists across all devices and API levels. The internal storage is inaccessible by the user under normal circumstances — exception to that is when the user has root privileges, e.g. on a rooted phone. This storage houses, among other things, files from apps, e.g. databases. These files are only accessible by the app that originially places them in the interal storage — neither user nor other apps can access them. Files are removed when the app they belong to is uninstalled. The external storage on the other hand is more public. Files placed here can be read and written by the user as well as other apps and they remain even after the app they orignated from is uninstalled. When working with the external storage it is important to check that it is not currently used as Universal Serial Bus (USB) storage by a computer the device is connected to. Apps have by default read and write access to the directory they are installed in on the internal storage, but to access the external storage the app requires that the user grants the app a specific permission. This permission needs to be declared in the app's manifest file, a XML file that every app must have. This file contains information required by the Android system to allow the app to run, such as the activities contained in the app and the permissions the app requires. Beginning in Android 6.0 (API level 23) apps targeting that Andoir version need to request and aquire dangerous permissions at run time (Android Developers 2016m), whereas before the app was given all permissions listed in its manifest upon agreeing to a dialog popup when installing the app. Dangerous permissions cover access to the user's private data or to affect areas where the user stores their data or data that other belong to other apps (Android Developers 2016m). Since the user can revoke permissions for apps at any given time the developer needs to take extra steps to keep the app running even when some features need to be disabled because of missing permissions.

# Chapter 6

# Implementation

*Google's IDE Android Studio 2.1.2 was used for the implementation of the Android app prototype targeting Android 6.0 Marshmallow (API level 23).*

## 6.1 Stitch symbols

There are different ways to display a stitch symbol in an Android app — this section will give an overview of the possibilities and the solution chosen for the prototype. Since the `View` class already defines a canvas object with dedicated functions for drawing image and text content in its `onDraw()` method, it offers a good starting point. To decide between using the image or text format for displaying stitch symbols, a further look into what each format entails is necessary.

Image content needs to be specified in an Android project in the res directory under the drawable directory. This directory contains the image resources of the app, the drawables — this applies for icons, custom images, and other image content, excempting the launcher icon of the app, which is located in the mipmap directory. One way to use stitch symbols in an Android app would be to add every symbol as a drawable resource, and then draw those resources to the canvas of a view. For this a drawable would be needed for each individual stitch symbol, as well as way to map these drawable files to values that can be efficiently stored in a JSON file. The usage of many drawables in an app would also lead to an increase in app size, resulting in longer download times and larger storage demands. Both are an inconvenience to the user and can be problematic on older devices with less powerful hardware, making the app unsusable in the worst

case scenario. The other option for displaying symbols is to create a custom True Type Font (TTF) with glyphs for stitch symbols that is applied to text in the app. This is the solution used in the prototype. It offers several advantages over using image resources. For one, when using drawables it might be necessary to include several versions of the same file to ensure that they are displayed correctly on devices with different screen densities (Android Developers 2016l). This is not needed for text with a custom font: the glyphs are defined by Beziér curves, which are correctly rendered by the system for the individual screen densities. The usage of a font also allows to write each stitch as a character, simplifying the process of saving a pattern to a JSON file. For OpenType fonts, which TTF belongs to, Microsoft recommends 64000 as the maximum number of glyphs a font should contain (Microsoft Corporation 2014). This allows for a plethora of stitch symbols. There are several knitting fonts available online, e.g. the Kauri Knits font by Kauri 2016 and the Knitter's Symbol font by Xenakis 1998. To ensure that the knitting symbol glyphs fit within the style of the grid and row format an example of a custom knitting font was created for this thesis by the author.

The open source program FontForge [1] was used creation of the custom TTF knitting font used in the protoype. The knitting font contains a selection of 16 stitch symbols whose glyphs were defined by the author herself. The glyphs and their corresponding UTF-8 characters can seen in **Figure 6.1**. The available symbols the knitting font offers as well as the corresponding symbol descriptions need to be defined as string arrays in the Constants class in the project.

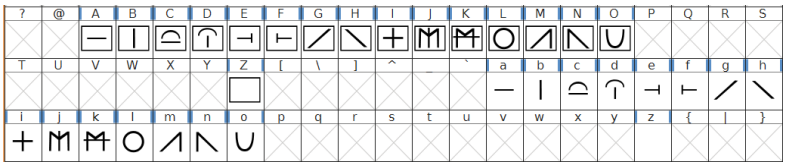



Figure 6.1: The custom knitting font used in the prototype

## 6.2 Pattern and Parsing between Pattern Formats

Using a custom font for the stitch symbols allows for presenting patterns as a combinations of characters. The actual pattern chart is saved to a JSON file as an `Array` of strings, where each string represents one row in the chart. For this a `Pattern` Plain Old

[1] `https://fontforge.github.io/en-US/` (last accessed: 08-10-2016)

Java Object (POJO) is used. It contains fields for the number of columns and rows, the current row set in the counter, and an array of strings, where each string represents a row in the pattern. A `Pattern` object's default state after initialization contains a pattern of the size 10 x 10 cells that is filled with the string representing an empty stitch — an empty stitch is represented by "Z"in the prototype. The class `Pattern` is a subclass of `Metadata.java`, a class containing fields for a pattern name and a Universally Unique Identifier (UUID). More about the `Metadata` class can be found in Section 6.3.

The actual pattern chart in the `Pattern` POJO is saved in the shortened row notation. To display the pattern in the grid and row format, the pattern needs to be parsed into forms that are usable by both formats. For that the class `PatternParser.java` is used. It converts between the array of strings used in the Pattern POJO, a two-dimensional `Array` of strings used in the `PatternGridView` (6.4.1), and a single string with linefeeds used in the row format's `Edittext` widget (6.4.2). For a more detailed explanation of the pattern notation forms refer to the corresponding sections.

## 6.3 Persistent Disk Storage

There are multiple ways persistent storage can be implemented in Android. A common choice is to use a `SQLite` database, which Android natively supports (Android Developers 2016k). Another choice is to save data in files. The prototype implements the latter, since it reduces the export of files to a simple matter of copying to a different directory. Therefore, instead of using a database, the protoype saves JSON files to the disk. This file type was chosen for the ease with which data can be saved and retrieved from the JSON file. Files can be saved either to internal or external storage, as explained in 5.2.5. Since permament accessibility of files cannot be ensured for files located on the external storage, the pattern files are saved to internal storage, in the default directory that Android allocates for the app. When exporting they are copied to a directory on the user-accessible external storage.

For this a `Pattern` POJO is serialized using Google's JSON library `Gson`[2], which handles the marshalling and unmarshalling of the files to Java objects and vice versa. `Gson` supports the usage of Java generics and can map JSON data to POJOs while maintaining inheritance hierarchies. The corresponding code can be in found the class

[2] `https://github.com/google/gson` (last accessed: 2016-08-09)

PatternStorage.java. ref here

The Pattern class inherits from the class Metadata which contains both a UUID which is used as a pattern's file name and the pattern name that is given by the user. When storing on disk, Pattern POJOs are converted to JSON using Gson and saved with the UUID as filename to avoid collisions in file names. Before that the metadata of the pattern is added to a local ArrayList of Metadatas in the PatternStorage class. This ArrayList in turn is marshalled to JSON and saved to the same directory as the pattern files. It acts as an index of all patterns saved on the device. Using this index file increases performance, since not all pattern files, with their potentially big pattern data, have to be accessed and unmarshalled — instead only the lightweight metadata files need to be loaded. Individual patterns can then be loaded using the UUID saved in the patterns Metadata.

## 6.4 Displaying a Pattern

### 6.4.1 Grid Format

When considering presenting data in a grid format, the most obvious solution is to first look at Android's own implementations of grids. Promising starting points for that are the TableLayout and the GridView class. After a brief investigation into the TableLayout, it quickly becomes apparent, that this layout is intended more as a way to position views, than to represent data in a grid. It can be likened to the HTML table tag (Android Developers 2016n), which is also used to position views within the constraints of cells, columns, and rows.

Android's code class, on the other hand, is designed with notion of displaying data. Each cell represents one data entry in a collection of data, the displaying of which is handled by an Adapter class attached to the grid view. Android ships with a specialized adapter for lists [3], as well as a BaseAdapter that can be subclassed for a custom handling and presentation of data. While this is a fitting solution for displaying symbols in a grid, it does not meet the requirements this project sets for the grid format. It is required that the column and row numbers are displayed next to the grid as axes. These axes should scale and scroll together with the grid, but should not be scrolled outside the

[3] https://developer.android.com/reference/android/widget/ListAdapter.html (accessed: 11-08-2016)

visible area, since the user would not be able to know the cell position then. For one, the `GridView` class does not support frozen cells, columns, or rows — as far as the author of this thesis was able to research. If the column numbers were to be displayed in the first row of the grid, they would move offscreen upon scrolling the grid. A possible solution to this would to use text views to act as axes to the grid and to display the column and row numbers next to it. Problematic with this approach would be the fine-tuning required to math the visuals of the text view to that of the grid. Line height, text size and the synchronization with scrolls and zooms performed on the grid would need to match perfectly. Since this does not classify as intended behavior for these views, a cohesive UI cannot be guaranteed. Furthermore, Android does not offer two dimensional scroll on any view except its WebView[4] – the GridView class natively only supports vertical scroll.

Therefore, in order to fulfill all requirements it makes the most sense to create a custom implementation of a view that supports the display of text in a grid, two-dimensional scroll, zoom, and axes that stick to the view bounds. Google's sample project *Interactive Chart*[5] and the corresponding training path[6] present an implementation example and were used as a guideline for the implementation of the class `PatternGridView.java`. reference here This class keeps a reference to a two-dimensional array of strings which represents the pattern with its columns and rows. The grid and its axes are drawn by overriding the `onDraw()` method, calulating the postion of the correspoinding lines and numbers with the dimensions of the view and pre-defined, hardcoded values for cell width and margin. The current version of the `PatternGridView` uses a `SimpleOnGestureListener`[7] to compute two-dimensional dragging. Android defines two different scrolling types for views: dragging and flinging (Android Developers 2016e). Dragging is executed by draging a finger across the device's touchscreen and results in a moving of the view corresponding to the dragging direction and speed. This means, that no matter the velocity of the dragging gesture, the view will not keep moving once the user lifts the finger involved in the gesture. A fling will keep moving the view with a speed and duration exponential to the velocity of the fling gesture, where the duration is calculated by introducing a

[4] `https://developer.android.com/reference/android/webkit/WebView.html` (accessed: 11-08-2016)

[5] `https://developer.android.com/shareables/training/InteractiveChart.zip` (accessed: 11-08-2016) A digital copy of this project can also be found on the CD attached to this thesis.

[6] `https://developer.android.com/training/gestures/scale.html#drag` (accessed: 11-08-2016)

[7] `https://developer.android.com/reference/android/view/GestureDetector.SimpleOnGestureListener.html` (last accessed: 2016-08-11)

friction to the scroll. This gesture can also be describe as a swiping motion performed on the touchscreen. Even after the finger has been lifted of the screen the fling continues to execute until it either runs its course or is interrupted.

The current version of the `PatternGridView` implements a simple dragging gesture and does not support flinging as of yet. For this a variable of type `PointF`[8] is saved locally to represent the offset scrolled. It is initiliazed with the value (0,0) and updated on every motion event that is recognized as dragging. The necessary calculations for these updates are done by overriding the `onScroll()` method in a custom `SimpleOnGestureListener`. This method has access to the horizontal and vertical distance scrolled, which is substracted from the offset. This is done because the value of the distance is positive when dragging towards the point of origin (the top left corner) and negative when dragging away from it. Therefore, the canvas needs to be translated in the opposite distance. The offset is then clamped to minimum and maximum values, to ensure that the grid will never completely move offscreen:

$$offset_{min} = (0, 0)$$

$$offset_{max} = (width_{view} - width_{content} - 2 * margin,$$
$$height_{view} - height_{content} - 2 * margin)$$

where margin is the distance of the grid from the top and left view edges that is reserved for the axes text.

Similiarly to dragging, scaling is implemented with a `SimpleOnScaleGestureListener`[9] with is connected to the view's `onTouchEvent()`. The scaling factor is then clamped at pre-defined maximum and minimum values and saved in a variable. During the drawing operations of the grid, axes, and text the scaling factor is then used to compute and draw the scaled visuals. On user touch on the grid the touched cell is calculated from the pixel position of the touch event and the currently selected stitch string is saved to that position in the two-dimensional pattern array. Following this the view is invalidated[10] and re-drawn with the updated pattern.

[8] `https://developer.android.com/reference/android/graphics/PointF.html` (last accessed: 2016-08-11)

[9] `https://developer.android.com/reference/android/view/ScaleGestureDetector.SimpleOnScaleGestureListener.html` (last accessed: 2016-08-11)

[10] `https://developer.android.com/reference/android/view/View.html#invalidate()` (last accessed 2016-08-11)

### 6.4.2 Row Format

The row editor should display line numbers at the left side of the screen and keep them in that position during horizontal scrolling, so that they will not move offscreen. Additionally, it should support the standard text editor functions: text select, copy, cut, and paste. It should display text in multiple lines that do not wrap at the end of the screen, but continue offscreen until a newline is input and the content needs to be scrollable horizontally and vertically. Android's `Edittext` widget[11] fulfills most of these requirements. The `Edittext` widget inherits from the class `EditText` which is specialised for diplaying text content. It supports standard text editing funtions, can display multiple lines of text, and supports vertical scrolling. Unfortunately, it does not natively support text lines to continue offscreen — upon reaching the width of the widget the text is wrapped to the next line. Another problem is, that the widget always automatically triggers the showing of Android's on-screen keyboard when it receives focus, i.e. when the user taps the view to start text input.

One instance, when the on-screen keyboard, also called the soft input method (Android Developers 2016i), is shown, is when the acitvity's main windows has input focus (Android Developers 2016c). An activity receives input focus on activity start and resume when containing an `Edittext` widget in it's view hierarchy. This behavior can be suppressed by declaring the state of the soft input method in the manifest fileof the project as hidden (see *Listing 6.1*).

ref here

```
1  ...
2  <activity android:name=".EditorActivity"
3              android:windowSoftInputMode="stateAlwaysHidden"/>
4  ...
```

Listing 6.1: Declaring on-screen keyboard hidden in manifest file.

Interaction with an `Edittext` widget will also show the on-screen keyboard: the widget's `onClick()` and `onLongClick()` listeners trigger the soft input method. To avoid this the custom widget `KeyboardlessEditText`[12], written by Danial Goodwin, is used in the prototype. It offers the the same functions as a native Android `Edittext` widget, but will suppress the showing of the on-screen keyboard.

[11] `https://developer.android.com/reference/android/widget/EditText.html`
[12] `https://github.com/danialgoodwin/android-widget-keyboardless-edittext`

To improve the visiblity of the lines the `LinedEditor`class is subclassed from the ref keyboardless `Edittext` widget and a background is drawn behind every other line. The `LinedEditor` is part of the `RowLinearLayout`, a custom `LinearLayout` which houses ref the complete row format editing functionality. This layout will be referred to as the row editor. A `LineNumberTextview`, a custom `TextView`, is placed to the left of the ref `LinedEditor` and displays the line numbers. To achieve a consistent look of both line numbers and editor text the same font and text size are set on both `EditText` and `Edittext` widget. To suppress the `Edittext` widget's line wrap behavior its width is set to the value *wrap˙content* (see *Listing 6.2*). This allows the widget to increase its width when text is added that exceeds the current width of the view.

```
1  ...
2    <de.muffinworks.knittingapp.views.LinedEditorEditText
3    android:paddingRight="50dp"
4    style="@style/DisplayEditTextStyle"
5    android:id="@+id/row_editor_edit_text"
6    android:gravity="center_vertical|left"
7    android:textSize="@dimen/row_editor_default_text_size"
8    android:layout_width="wrap_content"
9    android:layout_height="wrap_content"/>
10 ...
```

Listing 6.2: Excerpt from row˙editor.xml

The `LinedEditor`'s parent layout has a `Scroller`[13] attached to it which it used to scroll its children. The calculations needed for the scrolling behavior of the parent layout were adapted from the class `TwoDScrollView`, written by Clark 2010.

The row editor in the current version of the prototyep does not completely fulfill all requirements. On row editor instantiation the editor requests the `LinedEditor`'s dimensions and line count before the widget has been completely built. This results in a line count return value of zero, no matter how many lines of text have been set in the widget initially. The line numbers in the row editor then display the lowest line number possible: one. Additionally, the parent layout is not able to enable scrolling for offscreen text content, since the child measurements are incorrect. This could be solved by requesting the `Edittext` widget's dimensions at a later time when all views

[13] `https://developer.android.com/reference/android/widget/Scroller.html`

and layouts have been built. At the time of writing the author has not detemined the correct timing yet. More research into the lifecycle of the widget is required.

Additionally, the timing of measurements are also the cause for the another issue. Upon adding a new line which would lie outside of the visible area, the row editor should scroll the new line into view. Instead only the line above the new line is scrolled into view. This might be caused by measuring the `Edittext` widget bevore its height is updated to include the height of the new line. To determine a solution to this further research is needed.

The `Edittext` widget also returns the wrong position when more text is added at the end of a line. Instead of calulating the cursor's new position by increasing the value of its x-coordinateas by the width of added text, the returned position is at the first character of the next line. This might be due to the suppressed line wrapping behavior of `Edittext` widget in multi-line mode. Despite returning an incorrect cursor position upon text change, the text and cursor are diplayed at the correct location and not wrapped to the following line. This presents a problem when scrolling to offscreen text changes at the of a line. Instead of scrolling to the end of the edited line, the scroll will move the beginning of the next line into view.

Lasty, the line numbers do not stay visible when the row editor is scrolled horizontally. This behavior might be achieved by attaching different scrollers to the `EditText` for the line numbers and the `Edittext` widget, but the time constraints of this thesis did not allow further research.

## 6.5 Keyboard

The row and the grid editor each use different symbol keyboards. The keys of the grid editor keyboard feature stitch symbols and a delete button. Keys can be toggled to an active state — only one key at a time can be active. While a key is active, touching the grid will lead to the string corresponding with the active key being added to the pattern at the location of the touched cell. Since empty cells contain the designated empty character "Z", deletion works in the same way as setting a symbol on the grid. The keyboard is implemented using a `Gridview`[14], a default Android component to display a collection of items in a grid with equal spacing between all items. The gridview also

[14] `https://developer.android.com/reference/android/widget/GridView.html`

by default supports vertical scrolling. A grid item consists of text set on a button of the class `KnittingFontButton`, a custom class extending Android's own `Button` widget. The custom button sets the knitting font to display its string title as a knitting symbol. Set on the button are a click listener and a long click listener. The click listener toggles the state of the key and on long click the description of the symbol displayed on the button is shown at the bottom of the screen.

In the row editor the keyboard is divided in three sections. One section contains the stitch symbols, one a number pad and one an enter and a backspace button. Pressing a key on either number pad or symbols section appends the corresponding string or number to the editor at the current position of the cursor. The enter and backspace button call the system's enter and backspace key events from Android's software keyboard and do therefore not require custom handling, but only to be forwarded to the editor.

The symbols sections is also a Gridview, although with less columns than in the grid editor. The numpad uses the `CalculatorPadLayout` from Rahul Parsani's Material Calculator project[15]. The `CalculatorPadLayout` takes a number of child views, in this case `KnittingFontButtons`, as well as arguments for row and column count. It then calculates the size of the child views, so that all are equal in size. This custom layout is used because it optimally arranges its children in the available space without scrolling. A `Gridview`, on the other hand, is built to dynamically accommodate data and possible data changes — it is only concerned with the number of columns the data views can be placed in, if the `Gridview` bounds are too small to display all rows they will automatically placed offscreen and the `Gridview` will become scrollable — an undesired and atypical behaviour for a number pad, in the author's opinion.

## 6.6 Viewer with Row Counter

The row counter and the viewer are implemented in the `ViewerActivity` class. The activity's layout file defines a container `FrameLayout`[16] for the pattern content and below that the row counter UI. On its creation the activity instantiates a `PatternGridView` and a `RowEditorLinearLayout` and sets the data of the viewed pattern on both. The view and the layout can then be switched out at runtime inside their container by adding and removing the required view whenever the user decides to switch between grid and

source code ref here

[15] `https://github.com/rahulparsani/material-calculator`
[16] `https://developer.android.com/reference/android/widget/FrameLayout.html`

row format. At the current version of the prototype the row format is still experiencing some issues: the line numbers are not correctly instantiated and the pattern, if larger than the screen, is not scrollable inside the viewer.

The row counter below the pattern features a display the current row number the user is at in the knitting pattern and two buttons: one for increasing the counter and one for decreasing. The current row is set on both pattern format views as well, but only indicated with a visual highlight in the grid format. The grid format also scrolls the current row into view whenever an increase or decrease happens and the current row is offscreen.

The actionbar contains the following action buttons

- Switch pattern formats
- Open glossary
- Scroll to current row
- Export pattern
- Reset row counter
- Edit pattern

where the last three buttons are located in the overflow section.

## 6.7 Editor

The class `EditorActivity.java`contains, just like the `ViewerActivity`, a `FrameLayout` ref here to programmatically add the grid and row editor fragments to and allow easy switching between the visible fragments. The actionbar contains the following action buttons

- Switch pattern editor formats
- Save
- Set grid size
- Export pattern

- open glossary
- Edit pattern name
- Delete pattern

where the last four buttons can be found in the overflow section. The action button to set the grid size is only shown when the pattern is being edited in the grid format.

Upon switching the pattern formats changes to the pattern are automatically saved and upon success a short message is shown to the user. When the user tries to exit the editor while there are still unsaved changes a dialog is shown offerent the user to save the changes or to discard them and close the editor. After a pattern is exported an info dialog is displayed the directory on the external storage that the file was exported to. The activity also handles the showing of dialog fragments to request user input and processes the results. The dialogs handled in the `EditorActivity` are the `GridSizeDialogFragment`, the `PatternDeleteDialogFragment`, and the `PatternNameDialogFragment`.

ref
ref
ref

### Editor Fragments

Each of the two editor formats has its own fragment that displays the appropriate keyboard for the selected format. The fragments handle the saving, loading, and updating of the pattern data as well as the keyboard events. The grid format fragment also displays the dialog for changing the grid size.

## 6.8 Pattern List

The prototype launches with the `PatternListActivity`that displays all files currently indexed in the `Metadata` file (see section 6.3). For that Android's `ListView`[17] component is used. For each file the pattern name, a button to edit (pencil icon), and a button to delete (trash can icon) the pattern are shown. The edit button opens the selected pattern in the `EditorActivity` and upon delete the `PatternDeleteDialogFragment` is shown.

ref

[17] `https://developer.android.com/reference/android/widget/ListView.html`

## 6.9 Glossary

Like the `PatternListActivity`, the `GlossaryActivity` also uses a `ListView`[18] to display the symbols and their descriptions. The symbols and their descriptions are taken from the `Constants` class.

[18] see footnote 17

# Chapter 7

# User Test

As stated in Chapter 1 the prototype's UI is evaluated by iterative UI testing. For this a user is given a device running the prototype and a set of tasks related to the features of the prototype. While the user executes these tasks their reactions while using the prototype are observed and instances of problems with the UI noted. The user is then asked to summarize their experience in regards how easily they were able to use the features of the protoype and what elements of the interface they were confused about.

Ideally, user tests of the UI design are performed throughout the development process of a product, but time permitted for only one set of user tests during the development of this thesis.

After the implementation of a working prototype in accordance with the requirements set at the beginning of this thesis, the interview participants were invited to test the prototype. One participant had no experience nor interest in using knitting pattern charts and was therefore not included in the prototype test. The participants were asked to execute tasks derived from the requirements inside the prototype app during which their reactions were observed, with the main goal of determining the usability of the two pattern chart formats. After the completion of the tasks the user's feedback concerning the prototype was discussed. Users were asked to note wether the prototype met the expections they expressed during the interview and where it failed to do so. The users were also asked to judge the overall usability — the ease of use of the features, as well as the unambiguousness of the user interface. All user tests were held separately without communication between the participants. The results of these user tests are summarized in the following paragraphs.

Both participants were able to create and name a new pattern without problems. When first confronted with the default pattern in the row editor both expressed initial confusion about the shortened row format. After a brief explanation on what the number and symbol combination signified in the row format and how it would be displayed expanded when viewed in the grid format both participants were quickly able to work with the row editor and accurately produce results they were aiming for. For this they at first relied on switching to the grid editor to see how their changes in the row editor would play out — they were only able to grasp entirely how the row editor functioned after seeing the changes they did to the pattern in the expanded grid format. The participants did not expect the editor to support the standard editor funtionalities such as selecting, copying, cutting and pasting text. Both participants had problems when asked for the first time to edit the pattern in the grid editor: they expected the same typing behaviour from the symbol keyboard that they encountered in the row editor. It took a few tries and an explanation of the toggling mechanics to enable them to successfully use the grid editor. In both tests the participants found and used the buttons to switch editors and save the pattern without problems. The same holds true for the menu entries to rename and delete the pattern currently open in the editor. At the time of testing, viewing the pattern in row format still had some bugs: larger patterns were not scrollable and the current row would not be highlighted upon increase or decrease of the row counter. The testing of the row viewer will there be disregarded for this user test iteration. All functions of the row counter and the grid viewer were used and understood by both participants from the start.

Following the instructed testing the participants were asked to express their feedback concerning the prototype. Both participants expressed the desire to see a tutorial or introduction to the formats upon first use of the app since it was not clear from the beginning that there were two formats available to present a pattern chart. After being faced with the row format on opening a pattern in the editor, the participants expected the grid editor kyeboard to behave like the one found in the row editor. They had problems understanding how to use the toggle symbols in the grid editor keyboard and were confused why nothing happend when they touched the grid. For the wish to have a symbol pre-selected was voiced, to indicate that symbols have to be selected to be set on the grid and to help the user understand that touching a cell leads to the selected symbol being set to that cell. One user also mentioned that the trash can icon for the button

designed to erase a cell was misleading, they expected the button to delete the whole pattern — an eraser icon would be better suited. Another problem was the missing background behind the line numbers in the grid editor, when the grid was scrolled it was hard to read the numbers. To improve this a solid background should be added to the line number sections.

The row viewer did not fulfill the participants' expectation at all — both agreed that at the very least the pattern needs to be scrollable and the current row should be indicated. The wish to set the current row in the viewer by tapping the current row number in the counter section was also expressed.

The participants agreed that the protoype met the expectations set by the preliminary interviews, except for the row viewer. They compared the process of creating, editing and viewing a pattern chart to their current methods, modifying a spreadsheet to take the form of a knitting pattern chart, and found the prototype to be much easier and efficient to use. They deemed the ability to edit and view a pattern in two formats very valuable, since same stitch repeptitions could be quickly entered in the row format without the hassle of entering every stitch individually in the grid editor. The grid editor offered the ability to easily view the whole pattern and to spot and correct mistakes in it, as well as input more varied stitches in a pattern. Both participants preferred the prototype to their current pattern editors and viewers. One participant expressed the wish for a prototype supporting the same functions with colors instead of stitch symbols for the creation and viewing of colored pattern charts.

The current version of the prototype does not implement the aforementioned improvements and changes yet.

# Chapter 8

# Evaluation and Discussion

This thesis looked at how a knitting pattern chart can be input and displayed on mobile Android devices, the findings of this were showcased in a working Android app prototype. This chapter will look at the current state of the prototype, compare that state to the requirements specified in the beginning, and summarize the issues encountered and insights gained throughout the development of this thesis.

The current version of the prototype supports the creation, deletion, editing and viewing of knitting pattern charts. The pattern charts are saved as JSON files in the apps directory in the internal storage. Pattern files can be imported into the app and exported to a default directory on the device's external storage. On app start all patterns indexed in the app are shown in a list. From that list a pattern can be selected to be viewed, to be opened inside the editor, or to be deleted. Buttons to import a pattern file or export all patterns are located at the top of the screen. Patterns are presented in two different formats, the row and the grid format, as described in chapter 4. While editing or viewing the user can switch at any time between the formats. While editing a pattern options for deletion, changing the pattern name and import are available as well. The viewer contains a row counter situated below the pattern chart and that display the current row number and buttons for increasing and decreasing. The grid format highlights the current row while being viewed and supports two-dimensional scroll and zooming.

Except for the viewing of patterns in row format, the prototype presents a working solution for the research goal stated in the beginning of this thesis. The requirements listed in Chapter 3.2 were met and the result of the first user tests positive. Known bugs

that exist in the current version of the prototype are listed below:

1. Imported files are not checked if they contain a pattern
2. Drawing of the grid needs to be improved: dimensions larger than 35 cause lag on interaction
3. No UI optimization for smaller screen sizes
4. Row editor needs to be improved (Scrolling in viewer, background for line numbers, highlight current row)
5. Symbol descriptions are German only

The biggest obstacles encountered during the development of this thesis were the implementation of two-dimensional scrolling in views and layouts and the `EditText` widget's native behavior. This concerns the showing of the on-screen keyboard, the constraints placed on its width and scrolling in multi-line mode, as discussed in section 6.4.2. The implementation of a custom scroller was a challenge due to the calculations necessary to ensure a clean, two-dimensional scrolling behavior that resembles the one found in Android's one-dimensional scrollers. Implementing a custom scroller takes time, as well as some trial and error, but presents in the end a solvable problem. The `EditText` issues on the other hand are not as easily resolved. Trying to override native widget behavior that is not meant to be changed is an awkward affair, and each API level has its own behaviors that need to be handled separately. Whether or not it might be possible to force the `EditText` widget into a working solution that meets the requirements set for this thesis consistently on different devices can at this point still not be answered. More research would be needed to determine an answer to this question. It might be that the best solution for this issue is the implementation of a custom text editor.

# Chapter 9

# Outlook

To create a first release version the prototype more user feedback would need to be implemented. Currently the prototype only shows a console message when an error occurs during the export or saving of a pattern and and user feedback for successful deletion and the changing of a pattern name is missing. Currently the prototype's UI is optimized for use on a Nexus 9 Android tablet, smaller screen sizes would need to be supported to guarantee a consistent UI across all devices.

Further features stated in the requirements and gathered from the user tests would be to support stitches wider than one cell for knitting techniques that span across multiple stitches, e.g. the instruction to knit two together (k2tog). The wish for a repeat counter next to the row counter has also been expressed during user tests, for cases where a certain pattern has to be repeated, e.g. as often done in scraves.

It is also possible to integrate the functions of the prototype into an app designed to manage everything connected to knitting projects. Such and app could allow the user to keep an inventory of all the needles and yarns in his possession, keep track of a shopping list for future projects and allow the input of written instructions. The option to add pictures to a pattern, either taken directly on the device or added from disk, as well as to share a pattern from inside the app, e.g. via E-mail or DropBox, would also fit well into such an app.

hello glossary should be here ):

# Abbreviations

**API** Application Programming Interface. 19, 24, 25, 42

**GUI** Graphical User Interface. 19

**IDE** Integrated Development Environment. 19

**JSON** JavaScript Object Notation. 14, 25–28, 41

**k2tog** knit two together. 43

**POJO** Plain Old Java Object. 26–28

**RS** right side. 3, 4

**SDK** Software Development Kit. 19

**TTF** True Type Font. 26

**UI** User Interface. 2, 13, 21, 23, 29, 34, 38, 42, 43

**USB** Universal Serial Bus. 24

**UUID** Universally Unique Identifier. 27, 28

**WS** wrong side. 4

**XML** Extensible Markup Language. 23, 24

# List of Figures

# Listings

# Bibliography


Android Developers. *Action Bar*. URL: https://developer.android.com/design/patterns/actionbar.html (visited on 08/09/2016).

— *Activities*. URL: https://developer.android.com/guide/components/activities.html (visited on 08/09/2016).

— *¡activity¿*. URL: https://developer.android.com/guide/topics/manifest/activity-element.html (visited on 08/09/2016).

— *Android, the world's most popular mobile platform*. URL: https://developer.android.com/about/android.html (visited on 08/09/2016).

— *Animating a Scroll Gesture*. URL: https://developer.android.com/training/gestures/scroll.html#term (visited on 08/09/2016).

— *Choose Internal or External Storage*. URL: https://developer.android.com/training/basics/data-storage/files.html#InternalVsExternalStorage (visited on 08/09/2016).

— *Creating event callbacks to the activity*. URL: https://developer.android.com/guide/components/fragments.html#EventCallbacks (visited on 08/09/2016).

— *Fragments*. URL: https://developer.android.com/guide/components/fragments.html (visited on 08/09/2016).

— *Handling Keyboard Input*. URL: https://developer.android.com/training/keyboard-input/index.html (visited on 08/09/2016).

— *Meet Android Studio*. URL: https://developer.android.com/studio/intro/index.html (visited on 08/09/2016).

— *Saving Data*. URL: https://developer.android.com/training/basics/data-storage/index.html (visited on 08/09/2016).

— *Supporting Multiple Screens*. URL: https://developer.android.com/guide/practices/screens_support.html (visited on 08/09/2016).

Android Developers. *System Permissions*. URL: https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous (visited on 08/09/2016).

— *Table*. URL: https://developer.android.com/guide/topics/ui/layout/grid.html (visited on 08/09/2016).

— *View*. URL: https://developer.android.com/reference/android/view/View.html (visited on 08/09/2016).

Association, Japan Knitting Certificate. *JIS L 0201-1995: Letter symbols for knitting stitch. Standard booklet*. Nov. 1995. URL: http://www.webstore.jsa.or.jp/webstore/Com/FlowControl.jsp?lang=en&bunsyoId=JIS+L+0201%3A1995&dantaiCd=JIS&status=1&pageNo=0 (visited on 04/04/2016).

Clark, Matt. *Android Two-Dimensional ScrollView*. June 2, 2010. URL: https://web.archive.org/web/20110625064025/http://blog.gorges.us/2010/06/android-two-dimensional-scrollview (visited on 08/09/2016).

Kauri. *Kauri's Knitting Font*. July 31, 2016. URL: https://sites.google.com/site/kauriknitsfont/ (visited on 08/09/2016).

knirirr. *BeeCount Knitting Counter*. May 8, 2016. URL: https://lh5.ggpht.com/CaLXmsrgU6JmB1iswLwffjY2eMfOgtt90H41RgHRmGPqro6XCrMdnkawc_TR4nhohYI=h900-rw (visited on 08/09/2016).

Lewis, Perri. *Pride in the wool: the rise of knitting*. July 6, 2011. URL: https://www.theguardian.com/lifeandstyle/2011/jul/06/wool-rise-knitting (visited on 08/09/2016).

Microsoft Corporation. *Recommendations for OpenType Fonts*. May 1, 2014. URL: https://www.microsoft.com/typography/otspec/recom.htm (visited on 08/09/2016).

Natter, Maria. *Stricken*. Niedernhausen: Falken Verlag, 1983.

Nielsen, Jakob. *Usability 101: Introduction to Usability*. Jan. 4, 2014. URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (visited on 08/09/2016).

Raz, Samuel. *Flat Knitting Technology*. Heidenheim: Druck Repo Verlag, 1993.

Warren, Jennifer K. *Knit tink— Row Counter*. Sept. 16, 2015. URL: https://lh5.ggpht.com/FfgMOtcw2WerHBjS1eqm8NsjuxnTcfPHvxenf-3hfC1NKsIGHp_SPhcQy07Zpru8AQ=h900-rw (visited on 04/04/2016).

Whitall, Jonathan. *The KnitML User's Guide*. Apr. 25, 2009. URL: `http://www.knitml.com/docs/users-guide.html#d0e246` (visited on 08/09/2016).

Xenakis, David. *Knitter's Symbols Fonts*. 1998. URL: `http://www.knittinguniverse.com/downloads/KFont/` (visited on 08/09/2016).

# Appendix A

# User Interviews

## A.1 Question catalogue

Q1. *Wie viele Stücke haben Sie im vergangenen Jahr gestrickt?*

Q2. *Für wen stricken Sie? Enkel, für sich selber?*

Q3. *Gibt es bestimmte Stücke, die Sie häufig Stricken?*

Q4. *Benutzen Sie bestimmte Techniken häufiger als andere?*

Q5. *Wie wählen Sie eine Strickmuster? Selber ausdenken, suchen (online))*

Q6. *Wie arbeiten Sie damit?*

Q7. *Wie gehen Sie vor wenn Sie mit einer Strickmusterschematik arbeiten?*

Q8. *Könnten Sie sich vorstellen ein Strickmuster von einem mobilen Gerät abzulesen?*

Q9. *In welchem Format würden Sie dies gerne sehen? (audial, visuell)*

Q10. *Wie würden Sie dem Gerät zu erkennen geben, dass eine Reihe fertig gestrickt wurde? (Sprachbefehl, Knopf drücken)*

Q11. *(Verschiedene Strickmustertemplates zeigen und nach der Lesbarkeit fragen)*

Q12. *Haben Sie schon einmal selber Strickmusterschematiken erstellt?*

Q13. *Könnten Sie sich vorstellen, dies auf einem Handy zu tun?*

Q14. *Wie würden Sie sich dabei die Eingabe vorstellen?*

Q15. *Bei welchen Aspekten des Stricken könnten Sie sich eine App als hilfreiche Unterstützung vorstellen*

## A.2 Interview with Thilo Ilg

A1. Hat das letzte mal 2009 gestrickt.

A2. Hat nur für Schule gestrickt, hatte 6 Jahre lang einen Strickkurs.

A3. Socken.

A4. Nadelspiel.

A5. Von Lehrkraft ausgesucht.

A6. Hat noch nicht mit Musterschematik gearbeitet.

A7. -

A8. Ja.

A9. Beides ok, bevorzugt visuell.

A10. Findet Spracheingabe sehr nützlich, Hände sind voll beim Stricken.

A11. Beide Ansichten sind gut, würde sich aber gestört fühlen bei breiten Mustern vom ständigen Scrollen und würde Landscape-Modus besser finden, da mehr Platz zum Lesen der Reihe.

A12. Nein.

A13. Ja.

A14. Würde gerne Bereiche markieren können für eine Masche. Hätte gerne Funktrion um mehrere Zellen zu markieren und dann mit einem Maschensymbol zu befüllen. Klicken zum auswählen einer Zelle, zB. zum Bearbeiten. wenn Zelle markiert, nach Eingabe eines Symbols soll dann gleich zur nächsten Zelle gesprungen werden.

A15. Würde gerne Bilder von dem fertigen Gestrickten sehen und schriftliche Anweisungen bevor er sich mit der Schematik befasst. Will für komplexe Muster auf jeden Fall Schematik haben, für simplere Muster eher nicht notwendig. Hätte gerne Symbole in verschiedenen Farben für Sichtbarkeit. Wünscht sich Knopf um auf aktuelle Reihe und default Zoomstufe zu springen.

## A.3 Interview with Nadine Kost

A1. 25.

A2. Freunde und für den Eigenbedarf.

A3. Fingerlose Handschuhe, Socken.

A4. Bevorzugt Rundstricken.

A5. Internet, würde gerne selber Muster schreiben, arbeitet am häufigsten mit schriftlichen Musteranweisungen.

A6. Keine besondere Arbeitsweise.

A7. Ausdrucken und mit einem Stift die vollendeten Reihen durchstreichen.

A8. Ja.

A9. Visuell.

A10. Würde gerne nach der Vollendung einer Reihe einen Knopf drücken können.Dies soll auch ausserhalb der App möglich sein, zum Beispiel wie in Spotify mit einem Eintrag in der Notification bar oder mit einem Lockscreen widget.

A11. Bevorzugt: Zeilenansicht mit Knopf für den Wechsel zwischen Zeilen- und Zellenansicht. Hätte gerne am Ende der Reihe die Anzahl der Maschen angezeigt.

A12. Nein.

A13. Ja, kann sich das besonders gut vorstellen für Farbmuster.

A14. Am Anfang sollte man die Grösse des Musters wählen können. In einem Raster dieser Grösse soll dann bei Tap auf eine Zelle eine Auswahlansicht eingeblendet

werden, aus der man Symbole für verschiedene Maschen wählen kann. Nach kurzer Überlegung: es wäre benutzerfreundlicher ein Symbol als aktiv zu kennzeichnen, welches dann bei Klick auf eine Zelle in diese eingetragen wird.

A15. Beim Reihenzählen in einer Strickmusterschematik. Projektmanagement für Strickprojekte. Als ein Übersetzer von metrischen Einheiten von Nadelgrössen, Gewichten und Längen in imperiale und umgekehrt. Hätte ebenfalls gerne schriftliche Anweisungen in einer App wo man mit Knopfdruck auf nächste Anweisung springen könnte, zB. in Verbindung mit Reihenzähler.

## A.4 Interview with Angela Thomas

A1. 49, das Meiste waren 72 einmal im Jahr. Strickt schon seit vielen Jahren, allerdings keine Muster(zB. Zopf) sondern nur Rechts-Links.

A2. Grösstenteils für Bekannte.

A3. Stulpen, Dreieckstücher.

A4. Nein.

A5. Internet, Strickzeitung, Muster durch Bekannte gelernt.

A6. Keine Erfahrung mit Musterstricken, hat bisher nur Häkelmuster (Form) benutzt.

A7. Für Häkelmuster: mit Stecknadel Reihe markieren.

A8. Ja.

A9. Hätte gerne eine Sprachausgabe der momentanen Reihe und würde diese dann durch Knopfdruck wieder wiederholen lassen.

A10. Sprachbefehl: durchaus denkbar.

A11. Beide Ansichten wurden als wichtig gefunden, ein Wechsel zB per Knopf ist sowohl bei der Mustererstellung als auch in der Strickansicht gewünscht. Zeilenansicht ist für Kurzschrift, Zellen zum genaueren Betrachten des Musters.

A12. Nein.

A13. Ja.

A14. In der Zeilenansicht, wobei dann zwischen Zeilen - und Zellenansicht gewechselt werden kann. Möchte nicht darauf achten zu müssen Zellen zu zählen, daher wird Zeileneingabe bevorzugt.

A15. Erklärung und anschauliches Beispiel für einzeilne Maschen beim Stricken denkbar, Strick-/Häkelmuster auf dem Gerät mitnehmen (hat selber keine Smartphone, könnte sich das aber vorstellen). Bevorzugt schriftliche Anweisungen bei Mustern und braucht Text um eine Musterschematik zu verstehen.

# Appendix B

# Source Code

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.muffinworks.knittingapp">

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".ViewerActivity"/>

        <activity android:name=".EditorActivity"
            android:windowSoftInputMode="stateAlwaysHidden"/>

        <activity android:name=".GlossaryActivity"/>

        <activity
            android:name=".PatternListActivity"
            android:label="Knitting App">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Listing B.1: AndroidManifest.xml

```java
package de.muffinworks.knittingapp;

import android.Manifest;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.pm.ActivityInfo;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.MenuItem;

import de.muffinworks.knittingapp.storage.PatternStorage;
import de.muffinworks.knittingapp.util.Constants;

public abstract class BaseActivity extends AppCompatActivity {

    protected String TAG = this.getClass().getSimpleName();

    protected PatternStorage mStorage;
    private ActionBar mActionBar;
    private AlertDialog mDialog;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mStorage = PatternStorage.getInstance();
        mStorage.init(this);

        mActionBar = getSupportActionBar();

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRAIT);
    }

    protected void enableBackInActionBar(boolean enabled) {
        mActionBar.setDisplayHomeAsUpEnabled(enabled);
        mActionBar.setDisplayShowHomeEnabled(enabled);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == android.R.id.home) {
            onBackPressed();
        }
        return super.onOptionsItemSelected(item);
    }

    protected void setActionBarTitle(String title) {
        mActionBar.setTitle(title);
    }

    protected boolean isExternalStoragePermissionGranted() {
        return checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE)
                == PackageManager.PERMISSION_GRANTED;
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (mDialog != null) mDialog.dismiss();
    }

    //https://developer.android.com/training/permissions/requesting.html
    protected void requestExternalStoragePermission() {
        if (!isExternalStoragePermissionGranted()) {
            if (shouldShowRequestPermissionRationale(
                    Manifest.permission.
```

```
                WRITE_EXTERNAL_STORAGE)) {
            // should show permission
            showAlertDialog(getString(R.string.
                info_storage_permission),
                new DialogInterface.OnClickListener
                    () {
                    @Override
                    public void onClick(
                        DialogInterface dialog, int
                        which) {
                        requestPermissions(new
                            String[]{Manifest.
                            permission.
                            WRITE_EXTERNAL_STORAGE},
                            Constants.
                                PERMISSION_REQUEST_WRITE_SD
                            );
                    }
                });
            return;
        }
        requestPermissions(new String[]{Manifest.
            permission.WRITE_EXTERNAL_STORAGE},
            Constants.
                PERMISSION_REQUEST_WRITE_SD);
    }
}
```

Listing B.2: BaseActivity.java

```
package de.muffinworks.knittingapp.util;

import android.os.Environment;

public final class Constants {

    public static final String EMPTY_SYMBOL = "Z";

    public static String KNITTING_FONT_PATH = "fonts/
        OwnKnittingFont.ttf";

    public static String METADATA_FILENAME = "metadata.
        json";

    public static final String EXPORT_DIR = "Knitting
        Patterns";

    //gets path to external storage that is user
        accessible and won't be deleted after app
        install
    public static final String EXPORT_FOLDER_PATH =
        Environment
            .getExternalStorageDirectory().
                getAbsolutePath()
            + "/" + EXPORT_DIR;

    public static int FILE_PICKER_REQUEST_CODE = 2342;

    public static final String EXTRA_PATTERN_ID = "de.
        muffinworks.EXTRA_PATTERN_ID";
    public static final String EXTRA_PATTERN_DELETED = "
        de.muffinworks.EXTRA_PATTERN_DELETED";

    public static final int PERMISSION_REQUEST_WRITE_SD
        = 1337;

    public static final int REQUEST_CODE_EDITOR = 1;

    public static final int DEFAULT_ROWS = 10;
    public static final int DEFAULT_COLUMNS = 10;

    public static final int MAX_ROWS_AND_COLUMNS_LIMIT =
        35;

    public static final String[] DEFAULT_PATTERN = {
            "10" + Constants.EMPTY_SYMBOL, "10" +
                Constants.EMPTY_SYMBOL,
            "10" + Constants.EMPTY_SYMBOL, "10" +
                Constants.EMPTY_SYMBOL,
            "10" + Constants.EMPTY_SYMBOL, "10" +
                Constants.EMPTY_SYMBOL,
            "10" + Constants.EMPTY_SYMBOL, "10" +
                Constants.EMPTY_SYMBOL,
            "10" + Constants.EMPTY_SYMBOL, "10" +
                Constants.EMPTY_SYMBOL
    };

    public static final String [] SYMBOL_DESCRIPTIONS =
        {
            "Rechte Masche", "Linke Masche", "
                Rechtsverschränkte Masche", "
                Linksverschränkte Masche",
            "Masche rechts abheben", "Masche links
                abheben", "2 rechts zusammen stricken","
            "2 links zusammen stricken", "Randmasche","
                1 rechte Masche aufnehmen",
            "1 linke Masche aufnehmen","Abbinden", "
                Rechts neigende Zunahme", "Links
                neigende Zunahme",
            "Umschlag", "Leerer Platzhalter"
    };

}
```

```
50      public static final String [] SYMBOLS = {
51              "a", "b", "c", "d", "e", "f", "g", "h", "i",
                    "j", "k", "l", "m", "n", "o", "Z"
52      };
53  }
```

Listing B.3: Constants.java

```
1   package de.muffinworks.knittingapp;
2
3   import android.app.Activity;
4   import android.content.DialogInterface;
5   import android.content.Intent;
6   import android.os.Bundle;
7   import android.support.annotation.Nullable;
8   import android.support.v4.app.FragmentManager;
9   import android.support.v4.app.FragmentTransaction;
10  import android.support.v7.app.AlertDialog;
11  import android.view.Menu;
12  import android.view.MenuItem;
13  import android.view.View;
14  import android.widget.Button;
15
16  import java.io.IOException;
17
18  import de.muffinworks.knittingapp.fragments.
        GridEditorFragment;
19  import de.muffinworks.knittingapp.fragments.
        GridSizeDialogFragment;
20  import de.muffinworks.knittingapp.fragments.
        PatternDeleteDialogFragment;
21  import de.muffinworks.knittingapp.fragments.
        PatternNameDialogFragment;
22  import de.muffinworks.knittingapp.fragments.
        RowEditorFragment;
23  import de.muffinworks.knittingapp.storage.models.Pattern
        ;
24  import de.muffinworks.knittingapp.util.Constants;
25
26  public class EditorActivity extends BaseActivity
27          implements PatternNameDialogFragment.
                OnPatternNameInteractionListener,
28                  PatternDeleteDialogFragment.
                        OnPatternDeleteInteractionListener
                    ,
29                  GridSizeDialogFragment.
                        OnGridSizeInteractionListener {
30
31      private FragmentManager mFragmentManager;
32      private RowEditorFragment mRowEditorFragment;
33      private GridEditorFragment mGridEditorFragment;
34      private int mFragmentContainer = R.id.
            fragment_container;
35      private MenuItem mMenuItemSetGridSize;
36
37      private Pattern mPattern;
38      private String mPatternId = null;
39
40      private boolean mWasEdited = false;
41
42      @Override
43      protected void onCreate(@Nullable Bundle
            savedInstanceState) {
44          super.onCreate(savedInstanceState);
45          setContentView(R.layout.activity_editor);
46
47          enableBackInActionBar(true);
48
49          mPatternId = getIntent().getStringExtra(
                Constants.EXTRA_PATTERN_ID);
50
51          if (mPatternId != null) {
52              mPattern = mStorage.load(mPatternId);
53              setActionBarTitle(mPattern.getName());
54          }
55
56          mRowEditorFragment = RowEditorFragment.
                getInstance(mPatternId);
57          mGridEditorFragment = GridEditorFragment.
                getInstance(mPatternId);
58
59          mFragmentManager = getSupportFragmentManager();
60          FragmentTransaction fm = mFragmentManager.
                beginTransaction();
61          fm.replace(mFragmentContainer,
                mGridEditorFragment);
62          fm.commit();
63      }
64
65      @Override
66      public boolean onCreateOptionsMenu(Menu menu) {
67          getMenuInflater().inflate(R.menu.menu_editor,
                menu);
68          mMenuItemSetGridSize = menu.findItem(R.id.
                set_size);
69          mMenuItemSetGridSize.setVisible(false);
70          return true;
71      }
72
73      @Override
74      public boolean onOptionsItemSelected(MenuItem item)
            {
75          int id = item.getItemId();
76          if (id == R.id.set_size) {
77              showSetSizeDialog();
78          } else if (id == R.id.delete_pattern) {
79              showDeletePatternDialog();
80          } else if (id == R.id.switch_editor) {
81              switchEditors();
82          } else if (id == R.id.edit_pattern_name) {
83              showEditNameDialog();
84          } else if (id == R.id.save_pattern) {
85              savePattern();
86          } else if (id == R.id.open_glossary) {
87              startActivity(new Intent(this,
                        GlossaryActivity.class));
88          } else if (id == R.id.export_pattern) {
89              exportPattern();
```

```
90          }
91          return super.onOptionsItemSelected(item);
92      }
93
94      private void exportPattern() {
95          try {
96              mStorage.export(mPatternId);
97              showAlertDialog(getString(R.string.
                    success_export_pattern, Constants.
                    EXPORT_DIR));
98          } catch (IOException e) {
99              e.printStackTrace();
100         }
101     }
102
103     @Override
104     public void onBackPressed() {
105         if (wasPatternEdited()) {
106             AlertDialog saveBeforeExitDialog = new
                    AlertDialog.Builder(this)
107                 .setTitle(getString(R.string.
                        dialog_title_pattern_save_changes))
108                 .setPositiveButton(R.string.dialog_yes,
                        new DialogInterface.OnClickListener
                        () {
109                         @Override
110                         public void onClick(DialogInterface
                                dialog, int which) {
111                             savePattern();
112                             setResult(Activity.RESULT_OK);
113                             finish();
114                         }
115                 })
116                 .setNegativeButton(R.string.dialog_no,
                        new DialogInterface.OnClickListener
                        () {
117                         @Override
118                         public void onClick(DialogInterface
                                dialog, int which) {
119                             dialog.cancel();
120                             setResult(!mWasEdited ? Activity
                                    .RESULT_CANCELED : Activity.
                                    RESULT_OK);
121                             finish();
122                         }
123                 })
124                 .create();
125             saveBeforeExitDialog.show();
126         } else {
127             setResult(!mWasEdited ? Activity.
                    RESULT_CANCELED : Activity.RESULT_OK);
128             super.onBackPressed();
129         }
130     }
131
132     private void switchEditors() {
133             savePattern();
134
135             FragmentTransaction fm = mFragmentManager.
                    beginTransaction();
136         if (mRowEditorFragment.isVisible()) {
137             fm.replace(mFragmentContainer,
                        mGridEditorFragment);
138             mMenuItemSetGridSize.setVisible(true);
139         } else {
140             fm.replace(mFragmentContainer,
                        mRowEditorFragment);
141             mMenuItemSetGridSize.setVisible(false);
142         }
143         fm.commit();
144     }
145
146     private boolean wasPatternEdited() {
147         if (mRowEditorFragment.isVisible()) {
148             return mRowEditorFragment.hasPatternChanged
                        ();
149         } else {
150             return mGridEditorFragment.hasPatternChanged
                        ();
151         }
152     }
153
154     private void savePattern() {
155         if (wasPatternEdited()) {
156             if (mRowEditorFragment.isVisible()) {
157                 mRowEditorFragment.savePattern();
158             } else {
159                 mGridEditorFragment.savePattern();
160             }
161             mWasEdited = true;
162             mPattern = mStorage.load(mPatternId);
163         }
164     }
165
166     @Override
167     public void onSetChartSize(int columns, int rows) {
168         mGridEditorFragment.setGridSize(columns, rows);
169         savePattern();
170     }
171
172     public void showSetSizeDialog() {
173         GridSizeDialogFragment dialog =
                GridSizeDialogFragment.newInstance(
174                     mPattern.getColumns(),
175                     mPattern.getRows());
176         dialog.show(mFragmentManager, getString(R.string
                .tag_dialog_fragment_grid_size));
177     }
178
179     private void showEditNameDialog() {
180         Pattern pattern = mStorage.load(mPatternId);
181         PatternNameDialogFragment dialog =
                PatternNameDialogFragment.newInstance(
                pattern.getName());
182         dialog.show(mFragmentManager, getString(R.string
                .tag_dialog_fragment_edit_name));
183     }
184
185     private void showDeletePatternDialog() {
186         PatternDeleteDialogFragment dialog =
                PatternDeleteDialogFragment.newInstance(
                mPattern.getName());
```

```
187             dialog.show(mFragmentManager, getString(R.string
                    .tag_dialog_fragment_delete_pattern));
188         }
189
190         private void refreshFragmentData() {
191             mGridEditorFragment.notifyDataChanged();
192             mRowEditorFragment.notifyDataChanged();
193         }
194
195         public void onNumPadClick(View view) {
196             String num = ((Button)view).getText().toString()
                    ;
197             mRowEditorFragment.onNumPadClick(num);
198         }
199
200         public void onDeleteToggled(View view) {
201             mGridEditorFragment.onDeleteToggled();
202         }
203
204         @Override
205         public void onSetName(String name) {
206             mPattern.setName(name);
207             mStorage.save(mPattern);
208             setActionBarTitle(mPattern.getName());
209             mWasEdited = true;
210             refreshFragmentData();
211         }
212
213         @Override
214         public void onConfirmDelete() {
215             mStorage.delete(mPatternId);
216             Intent resultIntent = new Intent();
217             resultIntent.putExtra(Constants.
                        EXTRA_PATTERN_DELETED, true);
218             setResult(Activity.RESULT_CANCELED, resultIntent
                        );
219             finish();
220         }
221     }
```

Listing B.4: EditorActivity.java

```
1   package de.muffinworks.knittingapp;
2
3   import android.os.Bundle;
4   import android.support.annotation.Nullable;
5   import android.support.v7.app.ActionBar;
6   import android.view.MenuItem;
7   import android.widget.ListView;
8
9   import de.muffinworks.knittingapp.views.adapters.
        GlossaryAdapter;
10
11  public class GlossaryActivity extends BaseActivity {
12
13      private ActionBar mActionBar;
14      private ListView mGlossaryListView;
15
16      @Override
17      protected void onCreate(@Nullable Bundle
            savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_glossary);
20
21          mActionBar = getSupportActionBar();
22          mActionBar.setDisplayHomeAsUpEnabled(true);
23          mActionBar.setDisplayShowHomeEnabled(true);
24          mActionBar.setTitle(R.string.
                activity_title_glossary);
25
26          mGlossaryListView = (ListView) findViewById(R.id
                .glossary_listview);
27          mGlossaryListView.setAdapter(new GlossaryAdapter
                (this));
28      }
29
30      @Override
31      public boolean onOptionsItemSelected(MenuItem item)
            {
32          if (item.getItemId() == android.R.id.home) {
33              onBackPressed();
34          }
35          return true;
36      }
37  }
```

Listing B.5: GlossaryActivity.java

```
1   package de.muffinworks.knittingapp.views.adapters;
2
3   import android.content.Context;
4   import android.graphics.Typeface;
5   import android.view.LayoutInflater;
6   import android.view.View;
7   import android.view.ViewGroup;
8   import android.widget.BaseAdapter;
9   import android.widget.TextView;
10
11  import de.muffinworks.knittingapp.R;
12  import de.muffinworks.knittingapp.util.Constants;
13
14  public class GlossaryAdapter extends BaseAdapter {
15
16      private LayoutInflater mInflator;
17      private Context mContext;
18
19      public GlossaryAdapter(Context context) {
```

```
20          mContext = context;
21          mInflator = (LayoutInflater) mContext.
                getSystemService(Context.
                LAYOUT_INFLATER_SERVICE);
22      }
23
24      @Override
25      public int getCount() {
26          return Constants.SYMBOLS.length;
27      }
28
29      @Override
30      public Object getItem(int position) {
31          return Constants.SYMBOLS[position];
32      }
33
34      @Override
35      public long getItemId(int position) {
36          return 0;
37      }
38
39      @Override
40      public boolean isEnabled(int position) {
41          return false;
42      }
43
44      @Override
45      public View getView(int position, View convertView,
            ViewGroup parent) {
46          GlossaryItemViewHolder viewHolder;
47
48          if (convertView == null) {
49              convertView = mInflator.inflate(R.layout.
                    view_item_glossary, null);
50              viewHolder = new GlossaryItemViewHolder(
                    convertView);
51              viewHolder.mSymbol.setTypeface(Typeface.
                    createFromAsset(mContext.getAssets(),
52                      Constants.KNITTING_FONT_PATH));
53              convertView.setTag(viewHolder);
54          } else {
55              viewHolder = (GlossaryItemViewHolder)
                    convertView.getTag();
56          }
57
58          viewHolder.mSymbol.setText(Constants.SYMBOLS[
                position]);
59          viewHolder.mDescription.setText(Constants.
                SYMBOL_DESCRIPTIONS[position]);
60
61          return convertView;
62      }
63
64      static class GlossaryItemViewHolder {
65          public TextView mSymbol;
66          public TextView mDescription;
67
68          public GlossaryItemViewHolder(View root) {
69              mSymbol = (TextView) root.findViewById(R.id.
                    symbol);
70              mDescription = (TextView) root.findViewById(
                    R.id.symbol_description);
71          }
72      }
73  }
```

Listing B.6: GlossaryAdapter.java

```
1   package de.muffinworks.knittingapp.fragments;
2
3   import android.os.Bundle;
4   import android.support.annotation.Nullable;
5   import android.support.design.widget.Snackbar;
6   import android.support.v4.app.Fragment;
7   import android.view.LayoutInflater;
8   import android.view.View;
9   import android.view.ViewGroup;
10  import android.widget.GridView;
11  import android.widget.LinearLayout;
12
13  import java.util.Arrays;
14
15  import de.muffinworks.knittingapp.R;
16  import de.muffinworks.knittingapp.storage.PatternStorage
        ;
17  import de.muffinworks.knittingapp.storage.models.Pattern
        ;
18  import de.muffinworks.knittingapp.views.PatternGridView;
19  import de.muffinworks.knittingapp.views.adapters.
        KeyboardToggleAdapter;
20
21  public class GridEditorFragment extends Fragment
22          implements KeyboardToggleAdapter.
                GridEditorKeyListener {
23
24      private static final String BUNDLE_ID = "id";
25
26      private PatternStorage mStorage;
27      private Pattern mPattern;
28      private PatternGridView mPatternGridView;
29      private GridView mKeyboard;
30      private LinearLayout mDeleteButtonContainer;
31      private KeyboardToggleAdapter mKeyboardAdapter;
32      private boolean mIsDeleteActive = false;
33
34      public static GridEditorFragment getInstance(String
            patternId) {
35          GridEditorFragment fragment = new
                GridEditorFragment();
36          if (patternId != null) {
37              Bundle bundle = new Bundle();
38              bundle.putString(BUNDLE_ID, patternId);
39              fragment.setArguments(bundle);
40          }
41          return fragment;
42      }
43
44      @Override
45      public void onCreate(@Nullable Bundle
```

```
        savedInstanceState) {
46          super.onCreate(savedInstanceState);
47          mStorage = PatternStorage.getInstance();
48          mStorage.init(getActivity());
49          if (getArguments() != null) {
50              mPattern = mStorage.load(getArguments().
                    getString(BUNDLE_ID));
51          }
52      }
53
54      @Nullable
55      @Override
56      public View onCreateView(LayoutInflater inflater,
            @Nullable ViewGroup container,
57                               @Nullable Bundle
                                     savedInstanceState) {
58          return inflater.inflate(R.layout.
                fragment_editor_grid, container, false);
59      }
60
61      @Override
62      public void onViewCreated(View view, @Nullable
            Bundle savedInstanceState) {
63          super.onViewCreated(view, savedInstanceState);
64          mPatternGridView = (PatternGridView) view.
                findViewById(R.id.grid);
65          if (mPattern != null){
66              mPatternGridView.setPattern(mPattern.
                    getPatternRows());
67          }
68          mDeleteButtonContainer = (LinearLayout) view.
                findViewById(R.id.
                grid_delete_button_container);
69          mKeyboard = (GridView) view.findViewById(R.id.
                keyboard_gridview);
70          mKeyboardAdapter = new KeyboardToggleAdapter(
                getActivity(), this);
71          mKeyboard.setAdapter(mKeyboardAdapter);
72          //set first key active
73          mKeyboardAdapter.getView(0, null, null).
                callOnClick();
74      }
75
76      public void notifyDataChanged() {
77          if (mPattern != null) {
78              mPattern = mStorage.load(mPattern.getId());
79              mPatternGridView.setPattern(mPattern.
                    getPatternRows());
80          }
81      }
82
83      public void savePattern() {
84          String[] newPatternRows = mPatternGridView.
                getPattern();
85          mPattern.setPatternRows(newPatternRows);
86          mStorage.save(mPattern);
87          Snackbar.make(
88                  getView(),
89                  R.string.success_save_pattern,
90                  Snackbar.LENGTH_SHORT).show();
91      }
92
93      public boolean hasPatternChanged() {
94          return !Arrays.deepEquals(mPatternGridView.
                getPattern(),mPattern.getPatternRows());
95      }
96
97      @Override
98      public void onKeyToggled(String key) {
99          setDeleteActive(false);
100         mPatternGridView.setSelectedKey(key);
101     }
102
103     public void onDeleteToggled() {
104         setDeleteActive(!mIsDeleteActive);
105     }
106
107     private void setDeleteActive(boolean active) {
108         mIsDeleteActive = active;
109         mKeyboardAdapter.setDeleteActive(mIsDeleteActive
                );
110         mPatternGridView.setDeleteActive();
111         if (mIsDeleteActive) {
112             mDeleteButtonContainer.setBackgroundColor(
                    getResources().getColor(R.color.red_400,
                     null));
113         } else {
114             mDeleteButtonContainer.setBackgroundColor(
                    getResources().getColor(R.color.
                    colorPrimary,
115                         null));
116         }
117     }
118
119     public void setGridSize(int columns, int rows) {
120         mPatternGridView.setGridSize(columns, rows);
121     }
122 }
```

Listing B.7: GridEditorFragment.java

```
1   package de.muffinworks.knittingapp.fragments;
2
3   import android.app.Dialog;
4   import android.content.Context;
5   import android.content.DialogInterface;
6   import android.os.Bundle;
7   import android.support.v4.app.DialogFragment;
8   import android.support.v7.app.AlertDialog;
9   import android.text.Editable;
10  import android.text.TextWatcher;
11  import android.view.View;
12  import android.widget.EditText;
```

```
13  import android.widget.LinearLayout;
14
15  import de.muffinworks.knittingapp.R;
16  import de.muffinworks.knittingapp.util.Constants;
17
18  public class GridSizeDialogFragment extends
        DialogFragment {
19
20      private static final String BUNDLE_COLUMNS = "
            columns";
21      private static final String BUNDLE_ROWS = "rows";
22
23      private int mColumns = 0;
24      private int mRows = 0;
25      private OnGridSizeInteractionListener mListener;
26
27      public GridSizeDialogFragment() {}
28
29      public static GridSizeDialogFragment newInstance(int
            columns, int rows) {
30          GridSizeDialogFragment fragment = new
                GridSizeDialogFragment();
31          Bundle args = new Bundle();
32          args.putInt(BUNDLE_COLUMNS, columns);
33          args.putInt(BUNDLE_ROWS, rows);
34          fragment.setArguments(args);
35          return fragment;
36      }
37
38      @Override
39      public void onCreate(Bundle savedInstanceState) {
40          super.onCreate(savedInstanceState);
41          if (getArguments() != null) {
42              mColumns = getArguments().getInt(
                    BUNDLE_COLUMNS);
43              mRows = getArguments().getInt(BUNDLE_ROWS);
44          }
45      }
46
47      @Override
48      public void onStart() {
49          super.onStart();
50          //http://stackoverflow.com/a/15619098/4738174
51          final AlertDialog dialog = (AlertDialog)
                getDialog();
52          if (dialog != null) {
53              (dialog.getButton(DialogInterface.
                    BUTTON_POSITIVE)).setOnClickListener(new
                     View.OnClickListener() {
54                  @Override
55                  public void onClick(View v) {
56                      int newColumns = Integer.parseInt(
                            mColumnsEdittext.getText().
                            toString());
57                      int newRows = Integer.parseInt(
                            mRowsEdittext.getText().toString
                            ());
58                      if (newColumns != mColumns ||
                            newRows != mRows) {
59                          onChartSizeSetResult(
60                                  Integer.parseInt(
                                        mColumnsEdittext.
                                        getText().toString()
                                        ),
61                                  Integer.parseInt(
                                        mRowsEdittext.
                                        getText().toString()
                                        )
62                          );
63                      }
64                      dismiss();
65                  }
66              });
67          }
68      }
69
70      private EditText mColumnsEdittext;
71      private EditText mRowsEdittext;
72
73      @Override
74      public Dialog onCreateDialog(Bundle
            savedInstanceState) {
75          LinearLayout content = (LinearLayout)
                getActivity().getLayoutInflater().
76                  inflate(R.layout.dialog_set_grid_size,
                        null);
77          mColumnsEdittext = (EditText) content.
                findViewById(R.id.edittext_columns);
78          mColumnsEdittext.setText(Integer.toString(
                mColumns));
79          mColumnsEdittext.addTextChangedListener(new
                DimensionTextWatcher(mColumnsEdittext,
                mColumns));
80          mColumnsEdittext.setSelection(mColumnsEdittext.
                length());
81
82          mRowsEdittext = (EditText) content.findViewById(
                R.id.edittext_rows);
83          mRowsEdittext.setText(Integer.toString(mRows));
84          mRowsEdittext.addTextChangedListener(new
                DimensionTextWatcher(mRowsEdittext, mRows));
85          mRowsEdittext.setSelection(mRowsEdittext.length
                ());
86
87          AlertDialog dialog = new AlertDialog.Builder(
                getActivity())
88                  .setPositiveButton(
89                          R.string.dialog_ok,
90                          new DialogInterface.
                                OnClickListener() {
91                              public void onClick(
                                    DialogInterface dialog,
                                    int id) {
92                                  //http://stackoverflow.
                                        com/a
                                        /15619098/4738174
93                                  //Do nothing here:
                                        button overriden in
                                        onStart()
94                                  //Empty onclick needed
                                        on older versions to
                                         instantiate button
95                              }
```

```
96                  })
97                  .setNegativeButton(
98                          R.string.dialog_cancel,
99                          new DialogInterface.
                                OnClickListener() {
100                             public void onClick(
                                    DialogInterface dialog,
                                    int id) {
101                                 // User cancelled the
                                        dialog
102                             }
103                 })
104                 .setTitle(getResources().getString(R.
                        string.dialog_title_grid_size))
105                 .setView(content)
106                 .create();
107         return dialog;
108     }
109
110     private void onChartSizeSetResult(int columns, int
            rows) {
111         mListener.onSetChartSize(columns, rows);
112         dismiss();
113     }
114
115     @Override
116     public void onAttach(Context context) {
117         super.onAttach(context);
118         if (context instanceof
                OnGridSizeInteractionListener) {
119             mListener = (OnGridSizeInteractionListener)
                    context;
120         } else {
121             throw new RuntimeException(context.toString
                    ()
122                     + getString(R.string.
                            error_must_implement_interface,
123                     "OnGridSizeInteractionListener"));
124         }
125     }
126
127     @Override
128     public void onDetach() {
129         super.onDetach();
130         mListener = null;
131     }
132
133     public interface OnGridSizeInteractionListener {
134         void onSetChartSize(int columns, int rows);
135     }
136
137     class DimensionTextWatcher implements TextWatcher {
138         private int oldValue;
139         private EditText editText;
140
141         public DimensionTextWatcher(EditText editText,
142                 int oldValue) {
143             this.editText = editText;
144             this.oldValue = oldValue;
145         }
146
147         @Override
            public void beforeTextChanged(CharSequence s,
                int start, int count, int after) {}
148
149         @Override
150         public void onTextChanged(CharSequence s, int
                start, int before, int count) {}
151
152         @Override
153         public void afterTextChanged(Editable s) {
154             if (s.length() == 0) {
155                 //no input
156                 ((AlertDialog)getDialog()).getButton(
                        AlertDialog.BUTTON_POSITIVE).
                        setEnabled(false);
157                 return;
158             } else if (Integer.parseInt(s.toString()) ==
                        0) {
159                 //input is 0
160                 editText.setError(getString(R.string.
                        error_dimension_zero));
161                 editText.setSelection(editText.length())
                        ;
162                 ((AlertDialog)getDialog()).getButton(
                        AlertDialog.BUTTON_POSITIVE).
                        setEnabled(false);
163             } else if (Integer.parseInt(s.toString()) >
                        Constants.MAX_ROWS_AND_COLUMNS_LIMIT) {
164                 //input > max dimens
165                 editText.setError(
166                         getString(R.string.
                                error_over_max_size,
                                Constants.
                                MAX_ROWS_AND_COLUMNS_LIMIT))
                                ;
167                 editText.setSelection(editText.length())
                        ;
168                 ((AlertDialog)getDialog()).getButton(
                        AlertDialog.BUTTON_POSITIVE).
                        setEnabled(false);
169             } else {
170                 ((AlertDialog)getDialog()).getButton(
                        AlertDialog.BUTTON_POSITIVE).
                        setEnabled(true);
171             }
172         }
173     }
174 }
```

Listing B.8: GridSizeDialogFragment.java

```java
package de.muffinworks.knittingapp.views.adapters;

import android.content.Context;
import android.support.design.widget.Snackbar;
import android.view.LayoutInflater;
import android.widget.BaseAdapter;

import de.muffinworks.knittingapp.util.Constants;

public abstract class KeyboardAdapterBase extends
    BaseAdapter {

    protected String [] mDescriptions;
    protected String[] mCharacters;
    protected static LayoutInflater inflater = null;
    protected Context mContext;
    protected Snackbar mSnackbar = null;

    public KeyboardAdapterBase(Context context) {

        mDescriptions = Constants.SYMBOL_DESCRIPTIONS;
        mCharacters = Constants.SYMBOLS;
        mContext = context;

        inflater = (LayoutInflater)context.
                getSystemService(Context.
                    LAYOUT_INFLATER_SERVICE);

    }

    @Override
    public int getCount() {
        return mCharacters.length;
    }

    @Override
    public Object getItem(int position) {
        return mCharacters[position];
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
}
```

Listing B.9: KeyboardAdapterBase.java

```java
package de.muffinworks.knittingapp.views.adapters;

import android.content.Context;
import android.support.design.widget.Snackbar;
import android.view.View;
import android.view.ViewGroup;

import de.muffinworks.knittingapp.R;
import de.muffinworks.knittingapp.views.
    KnittingFontButton;

public class KeyboardToggleAdapter extends
    KeyboardAdapterBase {

    public interface GridEditorKeyListener {
        void onKeyToggled(String key);
    }

    private int mActiveKeyPosition = -1;
    private GridEditorKeyListener mListener;


    public KeyboardToggleAdapter(Context context,
        GridEditorKeyListener listener) {
        super(context);
        mListener = listener;
    }

    public void setDeleteActive(boolean active) {
        if (active) {
            mActiveKeyPosition = -1;
            notifyDataSetChanged();
        }
    }

    @Override
    public View getView(final int position, View
        convertView, ViewGroup parent) {

        KnittingFontButton key;
        if (convertView == null) {
            key = (KnittingFontButton) inflater.inflate(
                R.layout.view_grid_key, null);
        } else {
            key = (KnittingFontButton) convertView;
        }

        key.setActive(mActiveKeyPosition == position);
        key.setText(mCharacters[position]);
        key.setOnLongClickListener(new View.
            OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                mSnackbar = Snackbar.make(v,
                    mDescriptions[position], Snackbar.
                    LENGTH_LONG)
                        .setAction(R.string.dialog_ok,
                            new View.OnClickListener() {
                            @Override
                            public void onClick(View v)
                                {
                                mSnackbar.dismiss();
                            }
                        });
                mSnackbar.show();
                return true;
            }
        });
        key.setOnClickListener(new View.OnClickListener
```

```
            () {
60              @Override
61              public void onClick(View v) {
62                  mActiveKeyPosition = position;
63                  mListener.onKeyToggled(mCharacters[
                        position]);
64                  notifyDataSetChanged();
65              }
```

```
1   package de.muffinworks.knittingapp.views.adapters;
2
3   import android.content.Context;
4   import android.support.design.widget.Snackbar;
5   import android.view.View;
6   import android.view.ViewGroup;
7
8   import de.muffinworks.knittingapp.R;
9   import de.muffinworks.knittingapp.views.
        KnittingFontButton;
10
11  public class KeyboardTypingAdapter extends
        KeyboardAdapterBase {
12
13      public interface RowEditorKeyListener {
14          void onKeyClicked(String key);
15      }
16
17      private RowEditorKeyListener mListener;
18
19      public KeyboardTypingAdapter(Context context,
            RowEditorKeyListener listener) {
20          super(context);
21          mListener = listener;
22      }
23
24      @Override
25      public View getView(final int position, View
            convertView, ViewGroup parent) {
26
27          KnittingFontButton key;
28          if (convertView == null) {
29              key = (KnittingFontButton) inflater.inflate(
                    R.layout.view_grid_key, null);
30          } else {
31              key = (KnittingFontButton) convertView;
32          }
```

```
1   package de.muffinworks.knittingapp.views;
2
3   import android.content.Context;
4   import android.graphics.Typeface;
5   import android.util.AttributeSet;
6   import android.widget.Button;
7
8   import de.muffinworks.knittingapp.R;
9   import de.muffinworks.knittingapp.util.Constants;
10
```

```
66          });
67
68          return key;
69      }
70  }
```

Listing B.10: KeyboardToggleAdapter.java

```
33
34          key.setText(mCharacters[position]);
35          key.setOnLongClickListener(new View.
                OnLongClickListener() {
36              @Override
37              public boolean onLongClick(View v) {
38                  mSnackbar = Snackbar.make(v,
                        mDescriptions[position], Snackbar.
                        LENGTH_LONG)
39                          .setAction(R.string.dialog_ok,
                                new View.OnClickListener() {
40                                @Override
41                                public void onClick(View v)
                                      {
42                                    mSnackbar.dismiss();
43                                }
44                            });
45                  mSnackbar.show();
46                  return true;
47              }
48          });
49          key.setOnClickListener(new View.OnClickListener
                () {
50              @Override
51              public void onClick(View v) {
52                  mListener.onKeyClicked(mCharacters[
                        position]);
53              }
54          });
55
56          return key;
57      }
58  }
```

Listing B.11: KeyboardTypingAdapter.java

```
11  public class KnittingFontButton extends Button {
12
13      public KnittingFontButton(Context context) {
14          super(context);
15          init(context);
16      }
17
18      public KnittingFontButton(Context context,
            AttributeSet attrs) {
```

```
19          super(context, attrs);
20          init(context);     }
21
22      public KnittingFontButton(Context context,
            AttributeSet attrs, int defStyleAttr) {
23          super(context, attrs, defStyleAttr);
24          init(context);     }
25
26      private void init(Context context) {
27          setTypeface(getKnittingTypeFace(context));
28      }
29
30      private Typeface getKnittingTypeFace(Context context
            ) {
31          return Typeface.createFromAsset(context.
                getAssets(), Constants.KNITTING_FONT_PATH);
32      }
33
34      public void setActive(boolean mIsActive) {
35          if (mIsActive) {
36              setTextColor(getResources().getColor(R.color
                    .colorPrimary, null));
37          } else {
38              setTextColor(getResources().getColor(R.color
                    .keyboard_button_text_color, null));
39          }
40      }
41  }
```

Listing B.12: KnittingFontButton.java

```
1   package de.muffinworks.knittingapp.views;
2
3   import android.content.Context;
4   import android.graphics.Point;
5   import android.graphics.Typeface;
6   import android.text.Layout;
7   import android.util.AttributeSet;
8
9   import net.simplyadvanced.widgets.KeyboardlessEditText2;
10
11  import de.muffinworks.knittingapp.R;
12  import de.muffinworks.knittingapp.util.Constants;
13
14  public class LinedEditorEditText extends
        KeyboardlessEditText2 {
15
16      public LinedEditorEditText(Context context,
            AttributeSet attrs) {
17          super(context, attrs);
18
19          //set cursor visible and to beginning and
                request input focus
20          //needed to update the parents size
21          setCursorVisible(true);
22          requestFocus();
23          setSelection(0);
24
25          float textSize = context.getResources().
                getDimension(R.dimen.
                row_editor_default_text_size);
26          setTypeface(Typeface.createFromAsset(context.
                getAssets(), Constants.KNITTING_FONT_PATH));
27      }
28
29      public Point getCursorPosition() {
30      //https://stackoverflow.com/questions/5044342/how-to
            -get-cursor-position-x-y-in-edittext-android
31          Layout layout = getLayout();
32          if (layout != null) { //check if called before
                layout is inflated
33              int pos = getSelectionStart();
34              int line = layout.getLineForOffset(pos);
35              int baseline = layout.getLineBaseline(line);
36              int bl = (int)layout.getPrimaryHorizontal(
                    pos);
37              Point test = new Point(bl, baseline);
38              return test;
39          }
40          return new Point(0,0);
41      }
42  }
```

Listing B.13: LinedEditorEditText.java

```
1   package de.muffinworks.knittingapp.views;
2
3   import android.content.Context;
4   import android.graphics.Typeface;
5   import android.util.AttributeSet;
6   import android.widget.TextView;
7
8   import de.muffinworks.knittingapp.util.Constants;
9
10  public class LineNumberTextView extends TextView {
11
12      private int lines = 0;
13
14      public LineNumberTextView(Context context) {
15          super(context);
16          setTypeface(Typeface.createFromAsset(context.
                getAssets(), Constants.KNITTING_FONT_PATH));
17      }
18
19      public LineNumberTextView(Context context,
            AttributeSet attrs) {
20          super(context, attrs);
```

```
21          setTypeface(Typeface.createFromAsset(context.
                getAssets(), Constants.KNITTING_FONT_PATH));
22      }
23
24      public void updateLineNumbers(int lineCount) {
25          lines = lineCount;
26          String linesString = "1";
27          for(int i = 1; i < lines; i++) {
28              linesString += "\n" + (i+1);
29          }
30          setText(linesString);
31      }
32
```

```
33      private int measureLineNumberTextWidth() {
34          return (int) getPaint().measureText(lines + "");
35      }
36
37      public int getExactWidth() {
38          return measureLineNumberTextWidth() +
                getPaddingRight() + getPaddingLeft();
39      }
40  }
```

Listing B.14: LineNumberTextView.java

```
1   package de.muffinworks.knittingapp.storage.models;
2
3   import java.util.UUID;
4
5   public class Metadata implements Comparable<Metadata> {
6       /**
7        * Used to identify the file this pattern is stored
             in.
8        */
9       private UUID id;
10      protected String name = "Default name";
11
12      public Metadata() {
13          id = UUID.randomUUID();
14      }
15
16      public String getFilename() {
17          return id + ".json";
18      }
19
20      public String getId() {
21          return id.toString();
22      }
23
24      public String getName() {
25          return name;
26      }
27
28      public void setName(String name) {
29          this.name = name;
30      }
31
32      public Metadata clone() {
33          Metadata m2 = new Metadata();
34          m2.id = id;
35          m2.name = name;
36          return m2;
37      }
38
39      @Override
40      public int compareTo(Metadata that) {
41          return this.name.compareTo(that.name);
42      }
43  }
```

Listing B.15: Metadata.java

```
1   package de.muffinworks.knittingapp.storage.models;
2
3   import java.util.Arrays;
4   import java.util.Objects;
5
6   import de.muffinworks.knittingapp.util.Constants;
7   import de.muffinworks.knittingapp.util.PatternParser;
8
9   public class Pattern extends Metadata {
10
11      private String[] patternRows =  Constants.
            DEFAULT_PATTERN;
12      private int rows = Constants.DEFAULT_ROWS;
13      private int columns = Constants.DEFAULT_COLUMNS;
14      private int currentRow = 1;
15
16
17      public Pattern() {
18          super();
19      }
20
21      public String[] getPatternRows() {
22          return patternRows;
23      }
24
25      public void setPatternRows(String[] patternRows) {
26          this.patternRows = patternRows;
27          this.rows = patternRows.length;
28          this.columns = PatternParser.
                parseRowToGridFormat(patternRows[0]).length;
29      }
30
31      public int getRows() {
32          return rows;
33      }
34
```

```
35      public int getColumns() {
36          return columns;
37      }
38
39      public int getCurrentRow() {
40          return currentRow;
41      }
42
43      public void setCurrentRow(int currentRow) {
44          this.currentRow = currentRow;
45      }
46
47      @Override
48      public boolean equals(Object o) {
49          if (this == o) return true;
50          if (o == null || getClass() != o.getClass())
                return false;
51          Pattern pattern = (Pattern) o;
52          return rows == pattern.rows &&
53                  columns == pattern.columns &&
54                  currentRow == pattern.currentRow &&
55                  Arrays.equals(patternRows, pattern.
                        patternRows) &&
56                  name.equals(pattern.name);
57      }
58
59      @Override
60      public int hashCode() {
61          return Objects.hash(patternRows, rows, columns,
                currentRow, name);
62      }
63  }
```

Listing B.16: Pattern.java

```
1   package de.muffinworks.knittingapp.fragments;
2
3   import android.app.Dialog;
4   import android.content.Context;
5   import android.content.DialogInterface;
6   import android.os.Bundle;
7   import android.support.annotation.NonNull;
8   import android.support.annotation.Nullable;
9   import android.support.v4.app.DialogFragment;
10
11  import de.muffinworks.knittingapp.R;
12
13  public class PatternDeleteDialogFragment extends
        DialogFragment {
14
15      private static final String BUNDLE_NAME = "name";
16
17      private OnPatternDeleteInteractionListener mListener
            ;
18      private String mName = "";
19
20      public PatternDeleteDialogFragment() {}
21
22      public static PatternDeleteDialogFragment
            newInstance(String name) {
23          PatternDeleteDialogFragment fragment = new
                PatternDeleteDialogFragment();
24          Bundle args = new Bundle();
25          args.putString(BUNDLE_NAME, name);
26          fragment.setArguments(args);
27          return fragment;
28      }
29
30      @Override
31      public void onCreate(@Nullable Bundle
            savedInstanceState) {
32          super.onCreate(savedInstanceState);
33          if (getArguments() != null) {
34              mName = getArguments().getString(BUNDLE_NAME
                    );
35          }
36      }
37
38      @NonNull
39      @Override
40      public Dialog onCreateDialog(Bundle
            savedInstanceState) {
41          return new android.app.AlertDialog.Builder(
                getActivity())
42                  .setTitle(getString(R.string.
                        dialog_title_pattern_delete, mName))
43                  .setPositiveButton(R.string.dialog_yes,
                        new DialogInterface.OnClickListener
                        () {
44                      @Override
45                      public void onClick(DialogInterface
                            dialog, int which) {
46                          mListener.onConfirmDelete();
47                      }
48                  })
49                  .setNegativeButton(R.string.dialog_no,
                        new DialogInterface.OnClickListener
                        () {
50                      @Override
51                      public void onClick(DialogInterface
                            dialog, int which) {
52                          dialog.cancel();
53                      }
54                  })
55                  .create();
56      }
57
58      @Override
59      public void onAttach(Context context) {
60          super.onAttach(context);
61          if (context instanceof
                OnPatternDeleteInteractionListener) {
62              mListener = (
                    OnPatternDeleteInteractionListener)
                    context;
63          } else {
64              throw new RuntimeException(context.toString
```

```
                ()
65                  + getString(R.string.
                        error_must_implement_interface,"
66                  "OnPatternDeleteInteractionListener"
                        ));
67          }
68      }
69
70      @Override
71      public void onDetach() {
72          super.onDetach();
73          mListener = null;
74      }
75
76      public interface OnPatternDeleteInteractionListener
            {
77          void onConfirmDelete();
78      }
79  }
```

Listing B.17: PatternDeleteDialogFragment.java

```
1   package de.muffinworks.knittingapp.views;
2
3   import android.content.Context;
4   import android.graphics.Canvas;
5   import android.graphics.Color;
6   import android.graphics.Paint;
7   import android.graphics.PointF;
8   import android.graphics.RectF;
9   import android.graphics.Typeface;
10  import android.util.AttributeSet;
11  import android.view.GestureDetector;
12  import android.view.MotionEvent;
13  import android.view.ScaleGestureDetector;
14  import android.view.View;
15
16  import de.muffinworks.knittingapp.R;
17  import de.muffinworks.knittingapp.util.Constants;
18  import de.muffinworks.knittingapp.util.PatternParser;
19
20  public class PatternGridView extends View {
21
22
23      private static final String TAG = "GridEditorView";
24
25      private boolean canBeEdited = true;
26      private final float CELL_WIDTH = 100.0f;
27      private final float MARGIN = 40.0f;
28      private final float ZOOM_FACTOR_MIN = 0.5f;
29      private final float ZOOM_FACTOR_MAX = 2.0f;
30      private final float DEFAULT_SYMBOL_TEXTSIZE = 60.0f;
31
32      private int rows = Constants.DEFAULT_ROWS;
33      private int columns = Constants.DEFAULT_COLUMNS;
34      private String[][] symbols = new String[columns][
            rows];
35
36      /**
37       * represents the grid content
38       */
39      private RectF mContentRect = new RectF();
40      /**
41       * represents the visible area on the screen minus
              the padding
42       */
43      private RectF mCanvasRect = new RectF();
44      private RectF mRowHighlightRect = new RectF();
45
46      private Paint mRowHighlightPaint;
47      private Paint mGridPaint;
48      private Paint mLabelTextPaint;
49      private Paint mSymbolPaint;
50
51      private float mScaleFactor = 1f;
52      private ScaleGestureDetector mScaleGestureDetector;
53
54      private GestureDetector mGestureDetector;
55      private PointF mTranslationOffset = new PointF(0, 0)
            ;
56      private boolean hasScrolled = false;
57
58      private String mSelectedSymbol = null;
59      private int mCurrentRow = 0;
60
61
62      public PatternGridView(Context context) {
63          super(context);
64          init(context);
65      }
66
67      public PatternGridView(Context context, AttributeSet
             attrs) {
68          super(context, attrs);
69          init(context);
70      }
71
72      public PatternGridView(Context context, AttributeSet
             attrs, int defStyleAttr) {
73          super(context, attrs, defStyleAttr);
74          init(context);
75      }
76
77      private void init(Context context) {
78          initPaints();
79          updateContentRect();
80          mScaleGestureDetector = new ScaleGestureDetector
                (context, new GridScaleListener());
81          mGestureDetector = new GestureDetector(context,
                new GridGestureListener());
82      }
83
84      private void updateContentRect() {
85          mContentRect.set(
86                  MARGIN,
87                  MARGIN,
88                  MARGIN + columns * CELL_WIDTH *
                        mScaleFactor,
89                  MARGIN + rows * CELL_WIDTH *
```

```
                    mScaleFactor
90          );
91      }
92
93      private void initPaints() {
94          mRowHighlightPaint = new Paint();
95          mRowHighlightPaint.setStrokeWidth(1);
96          mRowHighlightPaint.setColor(getResources().
                getColor(R.color.highlight_current_row));
97          mRowHighlightPaint.setStyle(Paint.Style.FILL);
98
99          mGridPaint = new Paint();
100         mGridPaint.setStrokeWidth(1);
101         mGridPaint.setColor(Color.BLACK);
102         mGridPaint.setStyle(Paint.Style.STROKE);
103
104         mLabelTextPaint = new Paint();
105         mLabelTextPaint.setAntiAlias(true);
106         mLabelTextPaint.setTextAlign(Paint.Align.LEFT);
107         mLabelTextPaint.setTextSize(20);
108         mLabelTextPaint.setColor(Color.BLACK);
109
110         mSymbolPaint = new Paint();
111         mSymbolPaint.setAntiAlias(true);
112         mSymbolPaint.setTextAlign(Paint.Align.CENTER);
113         mSymbolPaint.setTextSize(DEFAULT_SYMBOL_TEXTSIZE
                );
114         Typeface knittingFont = Typeface.createFromAsset
                (getContext().getAssets(), Constants.
                KNITTING_FONT_PATH);
115         mSymbolPaint.setTypeface(knittingFont);
116     }
117
118     @Override
119     protected void onSizeChanged(int w, int h, int oldw,
            int oldh) {
120         super.onSizeChanged(w, h, oldw, oldh);
121         mCanvasRect.set(
122                 getPaddingLeft(),
123                 getPaddingTop(),
124                 getPaddingLeft() + w,
125                 getPaddingTop() + h
126         );
127         if (mCurrentRow != 0) {
128             scrollCurrentRowToCenter();
129         }
130     }
131
132     public void scrollCurrentRowToCenter() {
133         float currentRowTop = getPixelPositionTopForRow(
                mCurrentRow);
134         mTranslationOffset.set(
135                 mTranslationOffset.x,
136                 mCanvasRect.height()/2 - currentRowTop +
                        CELL_WIDTH
137         );
138         clampOffset();
139         invalidate();
140     }
141
142     public int getRows() {
143         return rows;
144     }
145
146     public void setGridSize(int columns, int rows) {
147         this.rows = rows;
148         this.columns = columns;
149         String[][] newSymbols = new String[columns][rows
                ];
150
151         //fill new array with data from old: data should
                persist in location, if new array is
                smaller
152         //than old, the data will be cut off and lost
153         if (rows > 0 && columns > 0) {
154             for (int c = 0; c < columns; c++) {
155                 for (int r = 0; r < rows; r++) {
156                     if (c < symbols.length && r <
                            symbols[0].length) {
157                         newSymbols[c][r] = symbols[c][r
                                ];
158                     } else {
159                         newSymbols[c][r] = Constants.
                                EMPTY_SYMBOL;
160                     }
161                 }
162             }
163         }
164         symbols = newSymbols;
165         updateContentRect();
166         invalidate();
167     }
168
169     public void setSymbol(int column, int row) {
170         if (row >= 0
171                 && row < rows
172                 && column >= 0
173                 && column < columns) {
174
175             symbols[column][row] = mSelectedSymbol;
176         }
177     }
178
179     public void setPattern(String[] patternRows) {
180         String[][] pattern = PatternParser.
                parsePojoToGridFormat(patternRows);
181         setGridSize(pattern.length, pattern[0].length);
182         symbols = pattern;
183         invalidate();
184     }
185
186     public String[] getPattern() {
187         return PatternParser.parseGridFormatToPojo(
                symbols);
188     }
189
190     public void setDeleteActive() {
191         mSelectedSymbol = Constants.EMPTY_SYMBOL;
192     }
193
194     public void setSelectedKey(String key) {
195         mSelectedSymbol = key;
196     }
```

```
197
198         public void setCurrentRow(int newCurrentRow) {
199             mCurrentRow = newCurrentRow;
200             if (mCanvasRect.height() != 0.0 && mCurrentRow >
                    0) {
201                 scrollCurrentRowToCenter();
202             }
203         }
204
205         public void setCanBeEdited(boolean editable) {
206             canBeEdited = editable;
207         }
208
209         @Override
210         public boolean onTouchEvent(MotionEvent event) {
211             //see https://stackoverflow.com/questions
                    /9965695/how-to-distinguish-between-move-and
                    -click-in-ontouchevent
212             //handle simple click tap
213             if (event.getAction() == MotionEvent.ACTION_UP
                    && canBeEdited) {
214                 if (mSelectedSymbol != null && !hasScrolled)
                        {
215                     float x = event.getX();
216                     float y = event.getY();
217                     int row = calculateRowFromValue(y);
218                     int column = calculateColumnFromValue(x)
                            ;
219
220                     setSymbol(column, row);
221                     postInvalidate();
222                 } else {
223                     hasScrolled = false;
224                 }
225                 return true;
226             }
227
228             boolean retVal = mScaleGestureDetector.
                    onTouchEvent(event);
229             retVal = mGestureDetector.onTouchEvent(event) ||
                     retVal;
230             return retVal || super.onTouchEvent(event);
231         }
232
233         private int calculateRowFromValue(float y) {
234             return  (int)((y - mTranslationOffset.y - MARGIN
                    ) / (CELL_WIDTH * mScaleFactor));
235         }
236
237         private int calculateColumnFromValue(float x) {
238             return (int)((x - mTranslationOffset.x - MARGIN)
                     / (CELL_WIDTH * mScaleFactor));
239         }
240
241         private PointF getCellCenter(int column, int row) {
242             return new PointF(
243                     MARGIN
244                             + column * CELL_WIDTH *
                                    mScaleFactor
245                             + CELL_WIDTH / 2 * mScaleFactor,
246                     MARGIN
247                             + row * CELL_WIDTH *
                                    mScaleFactor
248                             + CELL_WIDTH / 2 * mScaleFactor
249                             + ((int) Math.abs(mSymbolPaint.
                                    getFontMetrics().top))/2
250             );
251         }
252
253         private float getPixelPositionTopForRow(int row) {
254             return mContentRect.top + (row - 1) * CELL_WIDTH
                        * mScaleFactor;
255         }
256
257         private float getPixelPositionBottomForRow(int row)
                {
258             return mContentRect.top + row * CELL_WIDTH *
                    mScaleFactor;
259         }
260
261         @Override
262         protected void onDraw(Canvas canvas) {
263             super.onDraw(canvas);
264             canvas.save();
265             canvas.translate(mTranslationOffset.x,
                    mTranslationOffset.y);
266
267             if (mCurrentRow != 0) {
268                 mRowHighlightRect.set(
269                         mContentRect.left,
270                         getPixelPositionTopForRow(
                                mCurrentRow),
271                         mContentRect.left + columns *
                                CELL_WIDTH * mScaleFactor,
272                         getPixelPositionBottomForRow(
                                mCurrentRow)
273                 );
274                 canvas.drawRect(mRowHighlightRect,
                        mRowHighlightPaint);
275             }
276
277             drawGrid(canvas);
278             drawAxisLabels(canvas);
279             drawSymbols(canvas);
280             canvas.restore();
281         }
282
283         private void drawSymbols(Canvas canvas) {
284             for(int c = 0; c < columns; c++) {
285                 for (int r = 0; r < rows; r++) {
286                     String symbol = symbols[c][r];
287                     if (symbol != null) {
288                         mSymbolPaint.setTextSize(
                                DEFAULT_SYMBOL_TEXTSIZE *
                                mScaleFactor);
289                         PointF location = getCellCenter(c, r
                                );
290                         canvas.drawText(
291                                 symbol,
292                                 location.x,
293                                 location.y,
294                                 mSymbolPaint
295                         );
```

```
296                 }
297             }
298         }
299     }
300
301     private void drawGrid(Canvas canvas) {
302         if (rows > 0 && columns > 0) {
303             for (int i = 0; i < rows + 1; i++) {
304                 canvas.drawLine(
305                         MARGIN,
306                         (i * CELL_WIDTH * mScaleFactor)
                                + MARGIN,
307                         (columns * CELL_WIDTH *
                                mScaleFactor) + MARGIN,
308                         (i * CELL_WIDTH * mScaleFactor)
                                + MARGIN,
309                         mGridPaint);
310             }
311
312             for (int j = 0; j < columns + 1; j++) {
313                 canvas.drawLine(
314                         (j * CELL_WIDTH * mScaleFactor)
                                + MARGIN,
315                         MARGIN,
316                         (j * CELL_WIDTH * mScaleFactor)
                                + MARGIN,
317                         (rows * CELL_WIDTH *
                                mScaleFactor) + MARGIN,
318                         mGridPaint);
319             }
320         }
321     }
322
323     private void drawAxisLabels(Canvas canvas) {
324         //draw column labels
325         for (int r = 0; r < rows; r++) {
326             int text = r + 1;
327             canvas.drawText(
328                     text+"",
329                     5f  - mTranslationOffset.x, //text
                            width + offset
330                     (
331                             r * CELL_WIDTH *
                                    mScaleFactor
332                                     + CELL_WIDTH / 2 *
                                            mScaleFactor
333                                     + ((int) Math.abs(
                                            mLabelTextPaint.
                                            getFontMetrics()
                                            .top))/2
334                                     + MARGIN
335                     ),
336                     mLabelTextPaint);
337         }
338         //draw row labels
339         for (int c = 0; c < columns; c++) {
340             int text = c + 1;
341             canvas.drawText(
342                     text+"",
343                     (
344                             c * CELL_WIDTH *
345                                     mScaleFactor
346                                     + MARGIN
347                                     + CELL_WIDTH / 2 *
                                            mScaleFactor
348                                     - mLabelTextPaint.
                                            measureText(text
                                            + "") / 2
349                     ),
                        25f  - mTranslationOffset.y, //text
                            height + offset
350                     mLabelTextPaint);
351         }
352     }
353
354     public void resetZoom() {
355         mScaleFactor = 1.0f;
356         invalidate();
357     }
358
359     class GridGestureListener extends GestureDetector.
            SimpleOnGestureListener {
360
361         @Override
362         public boolean onScroll(MotionEvent e1,
                MotionEvent e2, float distanceX, float
                distanceY) {
363             //minus operation because scroll is inverse
                    to dragging
364             mTranslationOffset.x -= distanceX;
365             mTranslationOffset.y -= distanceY;
366
367             clampOffset();
368
369             hasScrolled = true;
370
371             postInvalidate();
372             return true;
373         }
374     }
375
376     private void clampOffset() {
377         float maxRightOffset = mCanvasRect.width() -
                mContentRect.width() - 2 * MARGIN;
378         float maxDownOffset = mCanvasRect.height() -
                mContentRect.height() - 2 * MARGIN;
379
380         if (mTranslationOffset.x > 0.0f ||
                maxRightOffset > 0) {
381             mTranslationOffset.x = 0.0f;
382         }
383         if (mTranslationOffset.y > 0.0f || maxDownOffset
                > 0) {
384             mTranslationOffset.y = 0.0f;
385         }
386         if (maxRightOffset < 0 && mTranslationOffset.x <
                maxRightOffset) {
387             mTranslationOffset.x = maxRightOffset;
388         }
389         if (maxDownOffset < 0 && mTranslationOffset.y <
                maxDownOffset) {
390             mTranslationOffset.y = maxDownOffset;
391         }
```

```
392         }
393
394         class GridScaleListener extends ScaleGestureDetector
                .SimpleOnScaleGestureListener {
395
396             private PointF viewportFocus = new PointF();
397             @Override
398             public boolean onScale(ScaleGestureDetector
                    detector) {
399
400                 mScaleFactor *= detector.getScaleFactor();
401                 mScaleFactor = Math.max(Math.min(
                        mScaleFactor, ZOOM_FACTOR_MAX),
                        ZOOM_FACTOR_MIN);
402
403                 updateContentRect();
404                 viewportFocus.set(
405                         detector.getFocusX(),
406                         detector.getFocusY()
407                 );
408                 invalidate();
409                 return true;
410             }
411         }
412     }
```

Listing B.18: PatternGridView.java

```
1   package de.muffinworks.knittingapp;
2
3   import android.content.DialogInterface;
4   import android.content.Intent;
5   import android.os.Bundle;
6   import android.support.annotation.Nullable;
7   import android.support.design.widget.
        FloatingActionButton;
8   import android.support.v4.app.FragmentManager;
9   import android.view.Menu;
10  import android.view.MenuItem;
11  import android.view.View;
12  import android.widget.ListView;
13
14  import com.google.gson.JsonSyntaxException;
15
16  import java.io.IOException;
17
18  import de.muffinworks.knittingapp.fragments.
        PatternNameDialogFragment;
19  import de.muffinworks.knittingapp.storage.models.Pattern
        ;
20  import de.muffinworks.knittingapp.util.Constants;
21  import de.muffinworks.knittingapp.views.adapters.
        PatternListAdapter;
22
23  public class PatternListActivity extends BaseActivity
24          implements PatternNameDialogFragment.
                OnPatternNameInteractionListener {
25
26      private ListView mPatternsList;
27      private PatternListAdapter mAdapter;
28      private FloatingActionButton mFab;
29
30      private MenuItem mExportAllMenu = null;
31
32      @Override
33      protected void onCreate(@Nullable Bundle
            savedInstanceState) {
34          super.onCreate(savedInstanceState);
35          setContentView(R.layout.activity_pattern_list);
36
37          enableBackInActionBar(false);
38
39          mPatternsList = (ListView) findViewById(R.id.
                patterns_list);
40          mAdapter = new PatternListAdapter(this);
41          mPatternsList.setAdapter(mAdapter);
42          mPatternsList.setItemsCanFocus(true);
43
44          mFab = (FloatingActionButton) findViewById(R.id.
                fab);
45          mFab.setOnClickListener(new View.OnClickListener
                () {
46              @Override
47              public void onClick(View v) {
48                  showSetNameDialog();
49              }
50          });
51
52          requestExternalStoragePermission();
53      }
54
55      @Override
56      protected void onResume() {
57          super.onResume();
58          mAdapter.notifyDataSetChanged();
59          checkExportAvailability();
60      }
61
62      @Override
63      public boolean onCreateOptionsMenu(Menu menu) {
64          getMenuInflater().inflate(R.menu.
                menu_pattern_list, menu);
65          mExportAllMenu = menu.findItem(R.id.export_all);
66          checkExportAvailability();
67          return super.onCreateOptionsMenu(menu);
68      }
69
70      private void checkExportAvailability() {
71          if (mExportAllMenu != null) {
72              mExportAllMenu.setVisible(mStorage.
                    listMetadataEntries().length > 0);
73          }
74      }
75
76      @Override
77      public boolean onOptionsItemSelected(MenuItem item)
            {
```

```
78          int id = item.getItemId();
79          if (id == R.id.import_pattern) {
80              if (isExternalStoragePermissionGranted()) {
81                  importFile();
82              } else {
83                  requestExternalStoragePermission();
84              }
85          } else if (id == R.id.export_all) {
86              if (isExternalStoragePermissionGranted()) {
87                  exportAllPatterns();
88              } else {
89                  requestExternalStoragePermission();
90              }
91          }
92          return super.onOptionsItemSelected(item);
93      }
94
95      private void exportAllPatterns() {
96          try {
97              mStorage.exportAll();
98              showAlertDialog(getString(R.string.
                    success_export_all, Constants.EXPORT_DIR
                    ));
99          } catch (IOException e) {
100             showAlertDialog(getString(R.string.
                    error_export));
101         }
102     }
103
104     private void importFile() {
105         Intent intent = new Intent(Intent.
                ACTION_GET_CONTENT);
106         intent.setType("application/json");
107         startActivityForResult(intent, Constants.
                FILE_PICKER_REQUEST_CODE);
108     }
109
110     @Override
111     protected void onActivityResult(int requestCode, int
            resultCode, Intent data) {
112         if (requestCode == Constants.
                FILE_PICKER_REQUEST_CODE) {
113             if (data != null) {
114                 try {
115                     final Pattern importedPattern =
                            mStorage.loadFromFile(data.
                            getData().getPath());
116                     if (mStorage.checkPatternDuplicate(
                            importedPattern)) {
117                         showAlertDialog(getString(R.
                                string.
                                info_import_pattern_already_exists
                                ),
118                                 new DialogInterface.
                                        OnClickListener() {
119                                     @Override
120                                     public void onClick(
                                            DialogInterface
                                            dialog, int
                                            which) {
121                                         mStorage.save(
                                                importedPattern
                                                );
122                                     }
123                                 });
124                     } else {
125                         mStorage.save(importedPattern);
126                     }
127                 } catch (JsonSyntaxException e) {
128                     showAlertDialog(getString(R.string.
                            error_import_no_json));
129                 }
130             }
131         }
132     }
133
134     private void showSetNameDialog() {
135         FragmentManager fm = getSupportFragmentManager()
                ;
136         PatternNameDialogFragment dialog =
                PatternNameDialogFragment.newInstance("");
137         dialog.show(fm, getString(R.string.
                tag_dialog_fragment_set_name));
138     }
139
140     @Override
141     public void onSetName(String name) {
142         Pattern pattern = new Pattern();
143         pattern.setName(name);
144         String patternId = pattern.getId();
145         mStorage.save(pattern);
146
147         Intent intent = new Intent(this, EditorActivity.
                class);
148         intent.putExtra(Constants.EXTRA_PATTERN_ID,
                patternId);
149         intent.putExtra(Constants.EXTRA_PATTERN_ID,
                patternId);
150         startActivity(intent);
151     }
152 }
```

Listing B.19: PatternListActivity.java

```
1   package de.muffinworks.knittingapp.views.adapters;
2
3   import android.app.AlertDialog;
4   import android.content.Context;
5   import android.content.DialogInterface;
6   import android.content.Intent;
7   import android.view.LayoutInflater;
8   import android.view.View;
9   import android.view.ViewGroup;
```

```
10  import android.widget.BaseAdapter;
11  import android.widget.ImageButton;
12  import android.widget.TextView;
13
14  import de.muffinworks.knittingapp.EditorActivity;
15  import de.muffinworks.knittingapp.R;
16  import de.muffinworks.knittingapp.ViewerActivity;
17  import de.muffinworks.knittingapp.storage.PatternStorage
        ;
18  import de.muffinworks.knittingapp.storage.models.
        Metadata;
19  import de.muffinworks.knittingapp.util.Constants;
20
21  public class PatternListAdapter extends BaseAdapter {
22
23      private Context mContext;
24      private Metadata[] mPatterns;
25      private PatternStorage mStorage = PatternStorage.
            getInstance();
26      private LayoutInflater mInflater;
27
28      public PatternListAdapter(Context context) {
29          mContext = context;
30          mStorage.init(mContext);
31          mPatterns = mStorage.listMetadataEntries();
32          mInflater = (LayoutInflater) mContext.
                getSystemService(Context.
                LAYOUT_INFLATER_SERVICE);
33      }
34
35      @Override
36      public int getCount() {
37          return mPatterns.length;
38      }
39
40      @Override
41      public Object getItem(int position) {
42          return mPatterns[position];
43      }
44
45      @Override
46      public long getItemId(int position) {
47          return position;
48      }
49
50      @Override
51      public void notifyDataSetChanged() {
52          mPatterns = mStorage.listMetadataEntries();
53          super.notifyDataSetChanged();
54      }
55
56      @Override
57      public View getView(final int position, View
            convertView, ViewGroup parent) {
58          PatternItemViewHolder viewHolder;
59
60          if (convertView == null) {
61              convertView = mInflater.inflate(R.layout.
                    view_item_list_pattern, null);
62              viewHolder = new PatternItemViewHolder(
                    convertView);
63              convertView.setTag(viewHolder);
64          } else {
65              viewHolder = (PatternItemViewHolder)
                    convertView.getTag();
66          }
67
68          viewHolder.mPatternName.setText(mPatterns[
                position].getName());
69          viewHolder.mEditButton.setOnClickListener(new
                View.OnClickListener() {
70              @Override
71              public void onClick(View v) {
72                  String patternId = ((Metadata)getItem(
                        position)).getId();
73                  Intent intent = new Intent(mContext,
                        EditorActivity.class);
74                  intent.putExtra(Constants.
                        EXTRA_PATTERN_ID, patternId);
75                  mContext.startActivity(intent);
76              }
77          });
78          viewHolder.mDeleteButton.setOnClickListener(new
                View.OnClickListener() {
79              @Override
80              public void onClick(View v) {
81                  Metadata m = (Metadata)getItem(position)
                        ;
82                  confirmDeleteDialog(m.getId(), m.getName
                        ());
83              }
84          });
85          convertView.setOnClickListener(new View.
                OnClickListener() {
86              @Override
87              public void onClick(View v) {
88                  String patternId = ((Metadata)getItem(
                        position)).getId();
89                  Intent intent = new Intent(mContext,
                        ViewerActivity.class);
90                  intent.putExtra(Constants.
                        EXTRA_PATTERN_ID, patternId);
91                  mContext.startActivity(intent);
92              }
93          });
94
95          return convertView;
96      }
97
98      public void confirmDeleteDialog(final String id,
            String name) {
99          AlertDialog dialog = new AlertDialog.Builder(
                mContext)
100                 .setTitle(mContext.getString(R.string.
                        dialog_title_pattern_delete, name))
101                 .setPositiveButton(R.string.dialog_ok,
102                         new DialogInterface.
                                OnClickListener() {
103                     @Override
104                     public void onClick(DialogInterface
                            dialog, int which) {
105                         PatternStorage storage =
                                PatternStorage.getInstance()
                                ;
```

```
106                         storage.delete(id);
107                         notifyDataSetChanged();
108                     }
109                 })
110                 .setNegativeButton(R.string.dialog_no,
111                         new DialogInterface.
                                OnClickListener() {
112                     @Override
113                     public void onClick(DialogInterface
                            dialog, int which) {
114                         dialog.cancel();
115                     }
116                 })
117                 .create();
118         dialog.show();
119     }
120
121     static class PatternItemViewHolder {
122         public TextView mPatternName;
123         public ImageButton mEditButton;
124         public ImageButton mDeleteButton;
125
126         public PatternItemViewHolder(View root) {
127             mPatternName = (TextView) root.findViewById(
                    R.id.pattern_name);
128             mEditButton = (ImageButton) root.
                    findViewById(R.id.button_edit);
129             mDeleteButton = (ImageButton) root.
                    findViewById(R.id.button_delete);
130         }
131     }
132 }
```

Listing B.20: PatternListAdapter.java

```
1   package de.muffinworks.knittingapp.fragments;
2
3   import android.app.Dialog;
4   import android.content.Context;
5   import android.content.DialogInterface;
6   import android.os.Bundle;
7   import android.support.annotation.NonNull;
8   import android.support.annotation.Nullable;
9   import android.support.v4.app.DialogFragment;
10  import android.support.v7.app.AlertDialog;
11  import android.text.Editable;
12  import android.text.InputFilter;
13  import android.text.Spanned;
14  import android.text.TextWatcher;
15  import android.widget.EditText;
16  import android.widget.LinearLayout;
17
18  import java.util.regex.Pattern;
19
20  import de.muffinworks.knittingapp.R;
21  import de.muffinworks.knittingapp.util.Constants;
22
23  public class PatternNameDialogFragment extends
        DialogFragment {
24
25      private static final String BUNDLE_NAME = "name";
26      private static final int MAX_NAME_LENGTH = 45;
27
28      private OnPatternNameInteractionListener mListener;
29      private String mName = "";
30
31      public PatternNameDialogFragment() {}
32
33      public static PatternNameDialogFragment newInstance(
            String name) {
34          PatternNameDialogFragment fragment = new
                PatternNameDialogFragment();
35          Bundle args = new Bundle();
36          args.putString(BUNDLE_NAME, name);
37          fragment.setArguments(args);
38          return fragment;
39      }
40
41      @Override
42      public void onCreate(@Nullable Bundle
            savedInstanceState) {
43          super.onCreate(savedInstanceState);
44          if (getArguments() != null) {
45              mName = getArguments().getString(BUNDLE_NAME
                        );
46          }
47      }
48
49      @NonNull
50      @Override
51      public Dialog onCreateDialog(Bundle
            savedInstanceState) {
52          final LinearLayout parent = (LinearLayout)
                getActivity().getLayoutInflater().
53                  inflate(R.layout.view_pattern_name_input
                            , null);
54          final EditText input = (EditText) parent.
                findViewById(R.id.input);
55          input.setText(mName);
56          input.setSelection(input.length());
57          input.setFilters(new InputFilter[]{new
                InputFilter.LengthFilter(MAX_NAME_LENGTH)});
58
59          final AlertDialog dialog = new AlertDialog.
                Builder(getActivity())
60                  .setPositiveButton(R.string.dialog_ok,
                        new DialogInterface.OnClickListener
                        () {
61                      public void onClick(DialogInterface
                            dialog, int id) {
62                          mListener.onSetName(input.
                                getText().toString());
63                      }
64                  })
65                  .setNegativeButton(R.string.
```

```
                    dialog_cancel, new DialogInterface.
                    OnClickListener() {
                    public void onClick(DialogInterface
                        dialog, int id) {
                        // User cancelled the dialog
                    }
                })
                .setTitle(getString(R.string.
                    dialog_title_pattern_name))
                .setView(parent)
                .create();

        input.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s
                , int start, int count, int after) {}

            @Override
            public void onTextChanged(CharSequence s,
                int start, int before, int count) {}

            @Override
            public void afterTextChanged(Editable s) {
                if (s.toString().isEmpty()) {
                    dialog.getButton(AlertDialog.
                        BUTTON_POSITIVE).setEnabled(
                        false);
                } else {
                    dialog.getButton(AlertDialog.
                        BUTTON_POSITIVE).setEnabled(true
                        );
                }
            }
        });
        dialog.setOnShowListener(new DialogInterface.
            OnShowListener() {
            @Override
            public void onShow(DialogInterface dialog) {
                ((AlertDialog)dialog).getButton(
                    AlertDialog.BUTTON_POSITIVE).
                    setEnabled(false);
            }
        });
        return dialog;
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof
            OnPatternNameInteractionListener) {
            mListener = (
                OnPatternNameInteractionListener)
                context;
        } else {
            throw new RuntimeException(context.toString
                ()
                    + getString(R.string.
                        error_must_implement_interface,
                    "OnPatternNameInteractionListener"))
                        ;
        }
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mListener = null;
    }

    public interface OnPatternNameInteractionListener {
        void onSetName(String name);
    }
}
```

Listing B.21: PatternNameDialogFragment.java

```
package de.muffinworks.knittingapp.util;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.regex.MatchResult;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PatternParser {

    private static final String EMPTY = "";
    private static final String LINEFEED = "\n";
    private static final String
        DEFAULT_EMPTY_PATTERN_STRING =
            "10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL
            +"10"+Constants.EMPTY_SYMBOL;
    private static final String
        REGEX_ALL_NUMBER_CHARACTER_PAIRS = "([0-9]*)([a-
        oA-OzZ])";
    private static final String
        REGEX_ALL_FORBIDDEN_CHARS = "[-+-,!@#$%^&*()
        ;/|<>\"':?= .]+|\\\\(?!n|r)";
    private static final String
        REGEX_LOOKBEHIND_LINEFEED = "(?<=\n)";
    private static final String REGEX_ALL_NON_DIGITS_END
        = "\\D+";
    private static final String REGEX_ALL_DIGITS_END = "
        \\d+$";
    private static final String REGEX_ALL_DIGITS = "[^\\
        D]";

```

```
31      static private Pattern pattern = Pattern.compile(
            REGEX_ALL_NUMBER_CHARACTER_PAIRS);
32
33      public static String parseGridToRowFormat(String[][]
            input) {
34          if (input == null || Arrays.deepEquals(input,
                new String[0][0])) return null;
35          String result = "";
36          for (int r = 0; r < input[0].length; r++) { //
                input[0] -> all rows have the same length
37
38              String previousSymbol = input[0][r]; //init
                    with first symbol in pattern
39              String currentSymbol = "";
40              int count = 0;
41
42              for (int c = 0; c < input.length; c++) {
43                  currentSymbol = input[c][r];
44                  if (currentSymbol.equals(previousSymbol)
                        ) {
45                      count++;
46                  } else {
47                      //append count and previousSymbol
48                      //save new previousSymbol
49                      //reset count
50                      result += count == 1 ?
                            previousSymbol : count +
                            previousSymbol;
51                      previousSymbol = currentSymbol;
52                      count = 1;
53                  }
54              }
55              result += count == 1 ? previousSymbol :
                    count + previousSymbol;
56              if (r != input[0].length - 1) {
57                  result += "\n";
58              }
59          }
60          //trim \n from end of string here
61          String test = result.substring(result.length() -
                1);
62          if ("\n".equals(test)) {
63              result = result.substring(0, result.length()
                    - 1);
64          }
65          return result;
66      }
67
68      public static String[][] parseRowToGridFormat(String
            input) {
69
70          if (input == null || input.isEmpty()) return
                null;
71
72          //expaned row of 3h -> hhh
73          ArrayList<String> expandedRows = new ArrayList
                <>();
74
75          //remove all characters that are not numeric or
                used for the symbols font
76          //https://stackoverflow.com/questions/1761051/
                difference-between-n-and-r
77          input = input.replaceAll(
                REGEX_ALL_FORBIDDEN_CHARS, EMPTY);
78
79          //split at linefeed \n in to get rows
80          String[] compressedRows = input.split(
                REGEX_LOOKBEHIND_LINEFEED);
81
82          int columns = 0;
83
84          for (int r = 0; r < compressedRows.length; r++)
                {
85              ArrayList<String> groupedSymbols = new
                    ArrayList<>(); //34g4gg111s -> 34g 4g g
                    111s
86
87              String row = compressedRows[r].replaceAll("
                    \\d+$", EMPTY);
88              row = row.replace("\n", "");
89
90              //check if row contained only \n and is now
                    ""
91              if (row.equals("" )) {
92                  row = Constants.EMPTY_SYMBOL;
93              }
94
95              Matcher m = pattern.matcher(row);
96              while(m.find()) {
97                  String group = m.group(0); //group 0 is
                        always entire match
98                  groupedSymbols.add(group);
99              }
100
101             //find out how many symbols are in the row
                    to get number of columns
102             int symbolCount = 0;
103             String expandedRow = "";
104
105             for (String group : groupedSymbols) {
106                 //remove all letters
107                 String symbolFactor = group.replaceAll(
                        REGEX_ALL_NON_DIGITS_END, EMPTY);
108                 //remove all numbers
109                 String symbol = group.replaceAll(
                        REGEX_ALL_DIGITS, EMPTY);
110
111                 if (!symbolFactor.isEmpty()) {
112                     //was grouped symbol e.g. 33k
113                     int factor = Integer.parseInt(
                            symbolFactor);
114
115                     if ((symbolCount + factor) >
                            Constants.
                            MAX_ROWS_AND_COLUMNS_LIMIT) {
116                         factor = Constants.
                                MAX_ROWS_AND_COLUMNS_LIMIT -
                                symbolCount;
117                     }
118
119                     symbolCount += factor;
120                     for (int j = 0; j < factor; j++) {
121                         expandedRow += symbol;
122                     }
123                 } else {
```

```
                    //only one symbol, not grouped e.g. k
                    symbolCount++;
                    expandedRow += symbol;
                }
            }
            if (symbolCount > columns) columns = symbolCount;
            expandedRows.add(expandedRow);
        }
        String[][] result = new String[columns][compressedRows.length];

        //itereate over String[][] and fill in from row strings
        for (int r = 0; r < compressedRows.length; r++) {
            for (int c = 0; c < expandedRows.get(r).length(); c++) {
                result[c][r] = Character.toString(expandedRows.get(r).charAt(c));
            }
        }
        //fill null places with placeholder for empty cell
        for (int c = 0; c < result.length; c++) {
            for (int r = 0; r < result[c].length; r++) {
                String symbol = result[c][r];
                if (symbol == null) {
                    result[c][r] = Constants.EMPTY_SYMBOL;
                }
            }
            String[] strings = result[c];
        }
        return result;
    }

    public static String parsePojoToRowFormat(String[] patternRows) {
        if (patternRows == null || patternRows.length == 0) return DEFAULT_EMPTY_PATTERN_STRING;
        String result = "";
        for (String row : patternRows) {
            result += row + "\n";
        }
        result = result.substring(0, result.length() - 1); //remove last \n
        return result;
    }

    public static String[] parseRowFormatToPojo(String rowInput) {
        //split after linefeed \n
        String[] rows = rowInput.split(REGEX_LOOKBEHIND_LINEFEED);
        int[] symbolsPerRows = new int[rows.length];
        int columns = 1;

        for (int r = 0; r < rows.length; r++) {
            ArrayList<MatchResult> groupedSymbols = new ArrayList<>();
            rows[r] = rows[r].replaceAll(REGEX_ALL_DIGITS_END, EMPTY);
            rows[r] = rows[r].replaceAll(LINEFEED, EMPTY);
            Matcher m = pattern.matcher(rows[r]);
            while(m.find()) {
                groupedSymbols.add(m.toMatchResult());
            }

            StringBuilder sb = new StringBuilder();

            int columnCount = 0;
            for (MatchResult group : groupedSymbols) {
                String symbolFactor = group.group(1);
                String symbol = group.group(2);
                if (columnCount >= Constants.MAX_ROWS_AND_COLUMNS_LIMIT)
                    break;

                if (!symbolFactor.isEmpty()) {
                    int factor = Integer.parseInt(symbolFactor);

                    if (factor + columnCount > Constants.MAX_ROWS_AND_COLUMNS_LIMIT) {
                        factor = Constants.MAX_ROWS_AND_COLUMNS_LIMIT - columnCount;
                    }

                    columnCount += factor;
                    symbolsPerRows[r] += factor;
                    sb.append(factor).append(symbol);
                } else {
                    columnCount++;
                    symbolsPerRows[r]++;
                    sb.append(symbol);
                }
            }
            rows[r] = sb.toString();
            if (columnCount > columns) columns = columnCount;
        }

        // Append trailing emtpy symbols to ensure the right amount of columns
        for (int r = 0; r < rows.length ; r++) {
            int diff = columns - symbolsPerRows[r];
            if (diff > 1) {
                rows[r] += diff + Constants.EMPTY_SYMBOL;
            } else if (diff == 1){
                rows[r] += Constants.EMPTY_SYMBOL;
            }
        }
        return rows;
    }

    public static String[] parseGridFormatToPojo(String[][] gridInput) {
        String rowFormat = parseGridToRowFormat(
```

```
            gridInput);
        return parseRowFormatToPojo(rowFormat);
    }

    public static String[][] parsePojoToGridFormat(
        String[] patternRows) {
        if (patternRows == null || patternRows.length ==
            0) {
            String[][] emptyDefaultPattern = new String[
                Constants.DEFAULT_COLUMNS][Constants.
                DEFAULT_ROWS];
            for (int c = 0; c < Constants.
                DEFAULT_COLUMNS; c++) {
                for (int r = 0; r < Constants.
                    DEFAULT_ROWS; r++) {
                    emptyDefaultPattern[c][r] =
                        Constants.EMPTY_SYMBOL;
                }
            }
            return emptyDefaultPattern;
        }
        String rowFormat = parsePojoToRowFormat(
            patternRows);
        return parseRowToGridFormat(rowFormat);
    }
}
```

Listing B.22: PatternParser.java

```
package de.muffinworks.knittingapp.util;

import org.junit.Test;

import java.util.ArrayList;
import java.util.Arrays;

import static org.junit.Assert.*;

public class PatternParserTest {

    @Test
    public void convertToGrid_2x2() {
        String in = "2h\n2y";
        String[][] expected = {
                {"h", "y"},
                {"h", "y"}
        };
        String[][] result = PatternParser.
            parseRowToGridFormat(in);
        assertTrue(Arrays.deepEquals(result, expected));
    }

    @Test
    public void convertToGrid_2x3() {
        String in = "2h\n3y";
        String[][] expected = {
                {"h", "y"},
                {"h", "y"},
                {".", "y"}
        };
        String[][] result = PatternParser.
            parseRowToGridFormat(in);
        assertTrue(Arrays.deepEquals(result, expected));
    }

    @Test
    public void convertToGrid_2() {
        String in = "2h";
        String[][] expected = {
                {"h"},
                {"h"}
        };
        String[][] result = PatternParser.
            parseRowToGridFormat(in);
        assertTrue(Arrays.deepEquals(result, expected));
    }

    @Test
    public void convertToGrid_02() {
        String in = "02h";
        String[][] expected = {
                {"h"},
                {"h"}
        };
        String[][] result = PatternParser.
            parseRowToGridFormat(in);
        assertTrue(Arrays.deepEquals(result, expected));
    }

    @Test
    public void convertToGrid_2_1() {
        String in = "2hh";
        String[][] expected = {
                {"h"},
                {"h"},
                {"h"}
        };
        String[][] result = PatternParser.
            parseRowToGridFormat(in);
        assertTrue(Arrays.deepEquals(result, expected));
    }

    @Test
    public void convertToGrid_2x2_empty() {
        String in = "2.\n2.";
        String[][] expected = {
                {".", "."},
                {".", "."}
        };
        String[][] result = PatternParser.
            parseRowToGridFormat(in);
        assertTrue(Arrays.deepEquals(result, expected));
    }
```

```
79
80      @Test
81      public void convertToGrid_emptyString() {
82          String in = "";
83          String[][] expected = null;
84          String[][] result = PatternParser.
                parseRowToGridFormat(in);
85          assertTrue(expected == result);
86      }
87
88      @Test
89      public void convertToGrid_null() {
90          String in = null;
91          String[][] expected = null;
92          String[][] result = PatternParser.
                parseRowToGridFormat(in);
93          assertTrue(expected == result);
94      }
95
96      @Test
97      public void convertToGrid_2_1_1x4() {
98          String in = "2hgt\n4d";
99          String[][] expected = {
100                 {"h", "d"},
101                 {"h", "d"},
102                 {"g", "d"},
103                 {"t", "d"}
104         };
105         String[][] result = PatternParser.
                parseRowToGridFormat(in);
106         assertTrue(Arrays.deepEquals(result, expected));
107     }
108
109     @Test
110     public void convertToGrid_3x2() {
111         String in = "2h\n2d\n2g";
112         String[][] expected = {
113                 {"h", "d", "g"},
114                 {"h", "d", "g"}
115         };
116         String[][] result = PatternParser.
                parseRowToGridFormat(in);
117         assertTrue(Arrays.deepEquals(result, expected));
118     }
119
120     @Test
121     public void convertToGrid_3x2_withForbiddenChars() {
122         String in = "2h\n2d\n_+-,!@#$%^&*();/|<>\"':?= 2
              g";
123         String[][] expected = {
124                 {"h", "d", "g"},
125                 {"h", "d", "g"}
126         };
127         String[][] result = PatternParser.
                parseRowToGridFormat(in);
128         assertTrue(Arrays.deepEquals(result, expected));
129     }
130
131     @Test
132     public void convertToGrid_2x3_test() {
133         String in = "hdg\nhdg\n";
134         String[][] expected = {
135                 {"h", "h"},
136                 {"d", "d"},
137                 {"g", "g"}
138         };
139         String[][] result = PatternParser.
                parseRowToGridFormat(in);
140         assertTrue(Arrays.deepEquals(result, expected));
141     }
142
143     @Test
144     public void convertToString_3x2() {
145         String[][] in = {
146                 {"h", "d", "g"},
147                 {"h", "d", "g"}
148         };
149         String expected = "2h\n2d\n2g";
150         String result = PatternParser.
                parseGridToRowFormat(in);
151         assertTrue(expected.equals(result));
152     }
153
154     @Test
155     public void
            convertToString_3x2WithEmptySpacesInTheEnd() {
156         String[][] in = {
157                 {"h", ".", "."},
158                 {"h", "d", "."}
159         };
160         String expected = "2h\n.d\n2.";
161         String result = PatternParser.
                parseGridToRowFormat(in);
162         assertTrue(expected.equals(result));
163     }
164
165     @Test
166     public void convertToString_3x2WithEmptySpaces() {
167         String[][] in = {
168                 {".", ".", "."},
169                 {".", ".", "."}
170         };
171         String expected = "2.\n2.\n2.";
172         String result = PatternParser.
                parseGridToRowFormat(in);
173         assertTrue(expected.equals(result));
174     }
175
176     @Test
177     public void convertToString_1x1() {
178         String[][] in = {
179                 {"."}
180         };
181         String expected = ".";
182         String result = PatternParser.
                parseGridToRowFormat(in);
183         assertTrue(expected.equals(result));
184     }
185
186     @Test
187     public void rowsWithSameLengthsToPojo() {
188         String in = "2n\n2g\n2h";
```

```
189         ArrayList<String> expected = new ArrayList<>();
190         expected.add("2n");
191         expected.add("2g");
192         expected.add("2h");
193         String[] result = PatternParser.
                parseRowFormatToPojo(in);
194         assertTrue(result.length == expected.size());
195         for (int i = 0; i < result.length; i++) {
196             if (!result[i].equals(expected.get(i))) fail
                    ();
197         }
198     }
199
200     @Test
201     public void rowsWithDifferentLengthsToPojo() {
202         String in = "2n\n4g\n2h";
203         ArrayList<String> expected = new ArrayList<>();
204         expected.add("2n2.");
205         expected.add("4g");
206         expected.add("2h2.");
207         String[] result = PatternParser.
                parseRowFormatToPojo(in);
208         assertTrue(result.length == expected.size());
209         for (int i = 0; i < result.length; i++) {
210             if (!result[i].equals(expected.get(i))) fail
                    ();
211         }
212     }
213
214     @Test
215     public void gridToPojo() {
216         String[][] in = {
217                 {"h", "f", "."},
218                 {"h", "g", "."},
219                 {"h", "g", "j"}
220         };
221         ArrayList<String> expected = new ArrayList<>();
222         expected.add("3h");
223         expected.add("f2g");
224         expected.add("2.j");
225         String[] result = PatternParser.
                parseGridFormatToPojo(in);
226         assertTrue(result.length == expected.size());
227
228         for (int i = 0; i < result.length; i++) {
229             if (!result[i].equals(expected.get(i))) fail
                    ();
230         }
231     }
232
233     @Test
234     public void pojoToGrid() {
235         String[][] expected = {
236                 {"h", "f", "."},
237                 {"h", "g", "."},
238                 {"h", "g", "j"}
239         };
240         String[] in = {
241                 "3h",
242                 "f2g",
243                 "2.j"
244         };
245         String[][] result = PatternParser.
                parsePojoToGridFormat(in);
246         assertTrue(Arrays.deepEquals(result, expected));
247     }
248
249     @Test
250     public void pojoToRow() {
251         String[] in = {
252                 "3h",
253                 "f2g",
254                 "2.j"
255         };
256         String expected = "3h\nf2g\n2.j";
257         String result = PatternParser.
                parsePojoToRowFormat(in);
258         assertTrue(result.equals(expected));
259     }
260
261     @Test
262     public void rowToPojoWithTrailingNumber() {
263         String in = "3r2";
264         String[] expected = {
265                 "3r"
266         };
267         String[] result = PatternParser.
                parseRowFormatToPojo(in);
268         assertTrue(Arrays.deepEquals(result, expected));
269     }
270
271     @Test
272     public void rowToGridWithTrailingNumber() {
273         String in = "3r2";
274         String[][] expected = {
275                 {"r"},
276                 {"r"},
277                 {"r"}
278         };
279         String[][] result = PatternParser.
                parseRowToGridFormat(in);
280         assertTrue(Arrays.deepEquals(result, expected));
281     }
282
283     @Test
284     public void rowToGridEmptyFirstRow() {
285         String in = "\n3h";
286         String[] expected = {
287                 "3.",
288                 "3h"
289         };
290         String[] result = PatternParser.
                parseRowFormatToPojo(in);
291         assertTrue(Arrays.deepEquals(result, expected));
292     }
293
294     @Test
295     public void rowToGridEmptyRowInMiddle() {
296         String in = "4f\n\n3h";
297         String[] expected = {
298                 "4f",
299                 "4.",
```

```
300                 "3h."
301             };
302             String[] result = PatternParser.
                    parseRowFormatToPojo(in);
303             assertTrue(Arrays.deepEquals(result, expected));
304         }
305
306         @Test
307         public void rowToGridEmptyRowAtEnd() {
308             String in = "4f\n\n3h\n";
309             String[][] expected = {
310                     {"f", ".", "h"},
311                     {"f", ".", "h"},
312                     {"f", ".", "h"},
313                     {"f", ".", "."}
314             };
315             String[][] result = PatternParser.
                    parseRowToGridFormat(in);
316             assertTrue(Arrays.deepEquals(result, expected));
317         }
318
319         @Test
320         public void rowToPojoEmptyRowAtEnd() {
321             String in = "\n4f\n\n3h\n";
322             String[] expected = {
323                     "4.",
324                     "4f",
325                     "4.",
326                     "3h."
327             };
328             String[] result = PatternParser.
                    parseRowFormatToPojo(in);
329             assertTrue(Arrays.deepEquals(result, expected));
330         }
331
332         @Test
333         public void rowToPojoEmptyRow() {
334             String in = "\n";
335             String[] expected = {
336                     "."
337             };
338             String[] result = PatternParser.
                    parseRowFormatToPojo(in);
339             assertTrue(Arrays.deepEquals(result, expected));
340         }
341
342         @Test
343         public void rowToGridEmptyRow() {
344             String in = "\n";
345             String[][] expected = {
346                     {"."}
347             };
348             String[][] result = PatternParser.
                    parseRowToGridFormat(in);
349             assertTrue(Arrays.deepEquals(result, expected));
350         }
351
352         @Test
353         public void rowToGridWidthEmptyRowInMiddle() {
354             String in = "3h\n\n4h";
355             String[][] expected = {
356                     {"h", ".", "h"},
357                     {"h", ".", "h"},
358                     {"h", ".", "h"},
359                     {".", ".", "h"}
360             };
361             String[][] result = PatternParser.
                    parseRowToGridFormat(in);
362             assertTrue(Arrays.deepEquals(result, expected));
363         }
364
365         @Test
366         public void rowToPojoMoreThanMaxColumns() {
367             String in = "55h3d";// + Constants.
                    MAX_ROWS_AND_COLUMNS_LIMIT + "h";
368             String[] expected = {
369                     "35h"
370             };
371             String[] result = PatternParser.
                    parseRowFormatToPojo(in);
372             assertTrue(Arrays.deepEquals(expected, result));
373         }
374
375         @Test
376         public void rowToGridMoreThanMaxColumns() {
377             String in = "55j5g";
378             int expected = Constants.
                    MAX_ROWS_AND_COLUMNS_LIMIT;
379             String[][] result = PatternParser.
                    parseRowToGridFormat(in);
380             assertTrue(expected >= result.length);
381         }
382 }
```

Listing B.23: PatternParserTest.java

```
1  package de.muffinworks.knittingapp.storage;
2
3  import android.content.Context;
4  import android.os.Environment;
5  import android.util.Log;
6
7  import com.google.gson.Gson;
8  import com.google.gson.JsonSyntaxException;
9  import com.google.gson.reflect.TypeToken;
10
11 import java.io.File;
12 import java.io.FileInputStream;
13 import java.io.FileNotFoundException;
14 import java.io.FileOutputStream;
15 import java.io.FileReader;
16 import java.io.FileWriter;
17 import java.io.IOException;
18 import java.lang.reflect.Type;
19 import java.nio.channels.FileChannel;
```

```
20  import java.util.Arrays;
21  import java.util.HashMap;
22  import java.util.List;
23
24  import de.muffinworks.knittingapp.R;
25  import de.muffinworks.knittingapp.storage.models.
        Metadata;
26  import de.muffinworks.knittingapp.storage.models.Pattern
        ;
27  import de.muffinworks.knittingapp.util.Constants;
28
29  public class PatternStorage {
30
31      private static final String TAG = "PatternStorage";
32
33      private Context mContext;
34      private Gson mGson = new Gson();
35      private HashMap<String, Metadata> mMetaDataTable;
36
37      private static PatternStorage storage = new
            PatternStorage();
38
39      public static PatternStorage getInstance() {
40          if (storage != null) {
41              return storage;
42          } else {
43              return new PatternStorage();
44          }
45      }
46
47      private PatternStorage() {}
48
49      public void init(Context context) {
50          this.mContext = context.getApplicationContext();
51          loadMetadata();
52      }
53
54      private String getApplicationDir() {
55          return mContext.getFilesDir().getPath();
56      }
57
58      private String getFilePathInApplicationDir(String
            fileName) {
59          return getApplicationDir() + "/" + fileName;
60      }
61
62      private boolean isExternalStorageWritable() {
63          return Environment.MEDIA_MOUNTED.equals(
                Environment.getExternalStorageState()) &&
64                  Environment.getExternalStorageDirectory
                        ().canWrite();
65      }
66
67      public void exportAll() throws IOException {
68          for (String id : mMetaDataTable.keySet()) {
69              export(id);
70          }
71      }
72
73      public File export(String id) throws IOException {
74          if(!isExternalStorageWritable())
75              throw new IOException(mContext.getString(R.
                        string.
                    error_external_storage_not_mounted));
76          File patternFile = new File(
                getFilePathInApplicationDir(id + ".json"));
77          File file = new File(Environment.
                getExternalStorageDirectory().
                getAbsolutePath()
                    + "/" + Constants.EXPORT_DIR);
78          file.mkdirs();
79          file = new File(file, id + ".json");
80          copyFile(patternFile, file);
81          return file;
82      }
83
84      public void importPattern(String path) throws
            JsonSyntaxException {
85          save(loadFromFile(path));
86      }
87
88      /**
89       * From https://stackoverflow.com/questions/9292954/
90           how-to-make-a-copy-of-a-file-in-android
91       */
92      private void copyFile(File src, File dst) throws
            IOException {
93          FileOutputStream outStream = new
                FileOutputStream(dst);
94          FileInputStream inStream = new FileInputStream(
                src);
95          FileChannel inChannel = inStream.getChannel();
96          FileChannel outChannel = outStream.getChannel();
97          inChannel.transferTo(0, inChannel.size(),
                outChannel);
98          inStream.close();
99          outStream.close();
100     }
101
102     private void loadMetadata() {
103         mMetaDataTable = new HashMap<>();
104
105         try {
106             File file = new File(getApplicationDir(),
                        Constants.METADATA_FILENAME);
107             FileReader fileReader = new FileReader(file)
                        ;
108             //https://sites.google.com/site/gson/gson-
                        user-guide#TOC-Collections-Examples
109             Type metadataLisType = new TypeToken<List<
                        Metadata>>(){}.getType();
110             List<Metadata> metadata = mGson.fromJson(
                        fileReader, metadataLisType);
111
112             try {
113                 fileReader.close();
114             } catch (IOException e) {
115                 logError(mContext.getString(R.string.
                            error_load_metadata));
116                 e.printStackTrace();
117             }
118             if (metadata != null) {
119                 for (Metadata m : metadata) {
120                     mMetaDataTable.put(m.getId(), m);
```

```
121                 }
122             }
123         } catch (FileNotFoundException e) {
124             mMetaDataTable = new HashMap<>();
125             return;
126         }
127     }
128
129     private void updateMetadata() {
130         try {
131             File file = new File(getApplicationDir(),
                        Constants.METADATA_FILENAME);
132             String json = mGson.toJson(mMetaDataTable.
                        values());
133             FileWriter fileWriter = null;
134             fileWriter = new FileWriter(file);
135             fileWriter.write(json);
136             fileWriter.close();
137         } catch (IOException e) {
138             logError(mContext.getString(R.string.
                        error_update_metadata));
139             e.printStackTrace();
140         }
141     }
142
143     public Metadata[] listMetadataEntries() {
144         Metadata[] m = mMetaDataTable.values().toArray(
                    new Metadata[mMetaDataTable.size()]);
145         if (m.length > 0) {
146             Arrays.sort(m);
147         }
148         return m;
149     }
150
151     public void save(Pattern pattern) {
152         try {
153             FileWriter fileWriter = new FileWriter(
                        getFilePathInApplicationDir(pattern.
                        getFilename()));
154             fileWriter.write(mGson.toJson(pattern));
155             fileWriter.close();
156             //call clone to put only metadata
                        information into hashmap, not actual
                        pattern related
157             //information -> needs less resources
158             mMetaDataTable.put(pattern.getId(), pattern.
                        clone());
159             updateMetadata();
160         } catch (IOException e) {
161             logError(mContext.getString(R.string.
                        error_save_pattern));
162             e.printStackTrace();
163         }
164     }
165
166     public Pattern loadFromFile(String path) throws
            JsonSyntaxException {
167         try {
168             FileReader reader = new FileReader(path);
169             return mGson.fromJson(reader, Pattern.class)
                        ;
170         } catch (FileNotFoundException e) {
171             logError(mContext.getString(R.string.
                        error_file_not_found, path));
172             return null;
173         }
174     }
175
176     public boolean checkPatternDuplicate(Pattern pattern
            ) {
177         String id = pattern.getId();
178         return mMetaDataTable.containsKey(id);
179     }
180
181     public Pattern load(String id) throws
            JsonSyntaxException {
182         return loadFromFile(getFilePathInApplicationDir(
                    id + ".json"));
183     }
184
185     public void clearAll() {
186         for (Metadata m : mMetaDataTable.values()) {
187             getFileFromApplicationDir(m.getFilename()).
                        delete();
188         }
189         getFileFromApplicationDir(Constants.
                    METADATA_FILENAME).delete();
190         mMetaDataTable.clear();
191     }
192
193     public void delete(Metadata pattern) {
194         getFileFromApplicationDir(pattern.getFilename())
                    .delete();
195         mMetaDataTable.remove(pattern.getId());
196         updateMetadata();
197     }
198
199     public void delete(String id) {
200         delete(mMetaDataTable.get(id));
201     }
202
203     private File getFileFromApplicationDir(String
            filename) {
204         return new File(getApplicationDir(), filename);
205     }
206
207     private void logError(String message) {
208         Log.e(TAG, message);
209     }
210 }
```

Listing B.24: PatternStorage.java

```java
package de.muffinworks.knittingapp.storage;

import android.content.Context;
import android.os.Environment;
import android.support.test.InstrumentationRegistry;
import android.support.test.runner.AndroidJUnit4;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

import java.io.File;
import java.io.IOException;

import de.muffinworks.knittingapp.storage.models.Metadata;
import de.muffinworks.knittingapp.storage.models.Pattern;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

@RunWith(AndroidJUnit4.class)
public class PatternStorageTest {

    private Context context = InstrumentationRegistry.getInstrumentation().getTargetContext().getApplicationContext();
    private PatternStorage storage;

    @Before
    public void setup() throws IOException {
        storage = PatternStorage.getInstance();
        storage.init(context);
        storage.clearAll();
    }

    @Test
    public void saveAndLoadPatternTest() throws IOException {
        assertNull(storage.load("fileIdThatDoesNotExist"));

        Pattern pattern = new Pattern();
        pattern.setName("scarf");
        pattern.setCurrentRow(2);
        pattern.setPatternRows(new String[]{
                "3e",
                "2er",
                "ttt",
                "3f",
                "3t"
        });
        assertEquals(3, pattern.getColumns());
        assertEquals(5, pattern.getRows());

        storage.save(pattern);
        File test = new File(context.getFilesDir().getPath().toString()+ "/"
                + pattern.getFilename());
        assertTrue(test.exists());

        Pattern p2 = storage.load(pattern.getId());
        assertEquals(pattern.equals(p2), true);
    }

    @Test
    public void listEntriesTest() throws IOException {
        assertTrue(storage.listMetadataEntries().length == 0);
        for(int i = 1; i <= 5; i++) {
            Pattern pattern = new Pattern();
            storage.save(pattern);
            assertTrue(storage.listMetadataEntries().length == i);
        }
    }

    @Test
    public void deleteTest() throws IOException {
        saveAndLoadPatternTest();
        PatternStorage storage2 = PatternStorage.getInstance();
        assertTrue(storage2.listMetadataEntries().length > 0);

        String id = storage2.listMetadataEntries()[0].getId();
        storage.delete(id);
        for (Metadata m : storage2.listMetadataEntries()) {
            if(m.getId().equals(id))
                fail();
        }
    }

    @Test
    public void exportTest() throws IOException {
        Pattern pattern = new Pattern();
        pattern.setName("scarf");
        pattern.setCurrentRow(2);
        pattern.setPatternRows(new String[]{
                "3e",
                "2er",
                "ttt",
                "3f",
                "3t"
        });
        storage.save(pattern);
        File file = storage.export(pattern.getId());
        assertTrue(file.exists());
    }

    @Test
    public void importTest() throws IOException {
        Pattern pattern = new Pattern();
        pattern.setName("scarf");
        pattern.setCurrentRow(2);
        pattern.setPatternRows(new String[]{
                "3e",
                "2er",
```

```
                "ttt",
                "3f",
                "3t"
        });
        storage.save(pattern);
        File file = storage.export(pattern.getId());
        assertTrue(file.exists());

        storage.delete(pattern.getId());
        storage.importPattern(file.getPath());
        Pattern pattern2 = storage.load(pattern.getId())
            ;
        assertEquals(pattern, pattern2);
    }

    @Test
    public void exportAllTest() throws IOException {
        storage.exportAll();
        for (Metadata md : storage.listMetadataEntries()
            ) {
            File file = new File(Environment.
                getExternalStoragePublicDirectory(
                Environment
                    .DIRECTORY_DOCUMENTS), md.getId() +
                        ".json");
            assertTrue(file.exists());
        }
    }

    @After
    public void cleanUp() throws IOException {
        storage.clearAll();
    }
}
```

Listing B.25: PatternStorageTest.java

```
package de.muffinworks.knittingapp.fragments;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.GridView;

import java.util.Arrays;

import de.muffinworks.knittingapp.R;
import de.muffinworks.knittingapp.layouts.
    RowEditorLinearLayout;
import de.muffinworks.knittingapp.storage.PatternStorage
    ;
import de.muffinworks.knittingapp.storage.models.Pattern
    ;
import de.muffinworks.knittingapp.views.adapters.
    KeyboardTypingAdapter;

public class RowEditorFragment extends Fragment
    implements KeyboardTypingAdapter.
    RowEditorKeyListener {

    private static final String TAG = "RowEditorFragment
        ";

    public RowEditorLinearLayout mRowEditorView;
    private Pattern mPattern;
    private PatternStorage mStorage;

    public static RowEditorFragment getInstance(String
        patternId) {
        RowEditorFragment fragment = new
            RowEditorFragment();
        if (patternId != null) {
            Bundle bundle = new Bundle();
            bundle.putString("id", patternId);
            fragment.setArguments(bundle);
        }
        return fragment;
    }

    @Override
    public void onCreate(@Nullable Bundle
        savedInstanceState) {
        super.onCreate(savedInstanceState);
        mStorage = PatternStorage.getInstance();
        mStorage.init(getActivity());
        if (getArguments() != null) {
            mPattern = mStorage.load(getArguments().
                getString("id"));
        }
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater,
        @Nullable ViewGroup container, @Nullable Bundle
        savedInstanceState) {
        return inflater.inflate(R.layout.
            fragment_editor_row, container, false);
    }

    @Override
    public void onViewCreated(View view, @Nullable
        Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mRowEditorView = (RowEditorLinearLayout) view.
            findViewById(R.id.row_editor_container);
        if (mPattern != null) {
            mRowEditorView.setPattern(mPattern.
                getPatternRows());
        }
        GridView mKeyboard = (GridView) view.
            findViewById(R.id.keyboard_gridview);
        mKeyboard.setAdapter(new KeyboardTypingAdapter(
            getActivity(), this));
```

```
63
64              view.findViewById(R.id.op_enter).
                    setOnClickListener(new View.OnClickListener
                    () {
65                  @Override
66                  public void onClick(View v) {
67                      onEnter();
68                  }
69              });
70
71              view.findViewById(R.id.op_delete).
                    setOnClickListener(new View.OnClickListener
                    () {
72                  @Override
73                  public void onClick(View v) {
74                      onDelete();
75                  }
76              });
77      }
78
79      public void savePattern() {
80          mPattern.setPatternRows(mRowEditorView.
                getPattern());
81          mStorage.save(mPattern);
82          Snackbar.make(getView(), getString(R.string.
                success_save_pattern), Snackbar.LENGTH_SHORT
                ).show();
83      }
84
85      public boolean hasPatternChanged() {
86          return !Arrays.deepEquals(mRowEditorView.
                getPattern(),mPattern.getPatternRows());
87      }
88
89      /**
90       * emulates the backspace key on the system's soft
             keyboard
91       */
92      public void onDelete() {
93          mRowEditorView.onDeletePressed();
94      }
95
96      /**
97       * emulates the enter key on the system's soft
             keyboard
98       */
99      public void onEnter() {
100         mRowEditorView.onEnterPressed();
101     }
102
103     public void onNumPadClick(String number) {
104         int start = mRowEditorView.getEditText().
                getSelectionStart();
105         mRowEditorView.getEditText().getText().insert(
                start, number);
106     }
107
108     public void notifyDataChanged() {
109         mPattern = mStorage.load(mPattern.getId());
110         mRowEditorView.setPattern(mPattern.
                getPatternRows());
111     }
112
113     @Override
114     public void onKeyClicked(String key) {
115         int start = mRowEditorView.getEditText().
                getSelectionStart();
116         mRowEditorView.getEditText().getText().insert(
                start, key.toUpperCase());
117     }
118 }
```

Listing B.26: RowEditorFragment.java

```
1   package de.muffinworks.knittingapp.layouts;
2
3   import android.content.Context;
4   import android.graphics.Point;
5   import android.graphics.PointF;
6   import android.graphics.Rect;
7   import android.support.design.widget.Snackbar;
8   import android.text.Editable;
9   import android.text.TextWatcher;
10  import android.util.AttributeSet;
11  import android.view.KeyEvent;
12  import android.view.LayoutInflater;
13  import android.view.MotionEvent;
14  import android.view.VelocityTracker;
15  import android.view.View;
16  import android.view.ViewConfiguration;
17  import android.view.ViewGroup;
18  import android.widget.EditText;
19  import android.widget.LinearLayout;
20  import android.widget.Scroller;
21
22  import de.muffinworks.knittingapp.R;
23  import de.muffinworks.knittingapp.util.Constants;
24  import de.muffinworks.knittingapp.util.PatternParser;
25  import de.muffinworks.knittingapp.views.
        LineNumberTextView;
26  import de.muffinworks.knittingapp.views.
        LinedEditorEditText;
27
28  public class RowEditorLinearLayout extends LinearLayout
        {
29
30      private static final String TAG = "
            RowEditorLinearLayout";
31
32      private LineNumberTextView lineNumbers;
33      private LinedEditorEditText editText;
34
35      private boolean mIsBeingDragged = false;
36      private Point mLastScrollTo = new Point();
37      private Scroller mScroller;
38      private PointF mLastMotion = new PointF();
39      private VelocityTracker mVelocityTracker;
```

```
40      private int mTouchSlop;
41      private int mMinimumVelocity;
42
43      public RowEditorLinearLayout(Context context) {
44          super(context);
45          init(context);
46      }
47
48      public RowEditorLinearLayout(Context context,
            AttributeSet attrs) {
49          super(context, attrs);
50          init(context);
51      }
52
53      public RowEditorLinearLayout(Context context,
            AttributeSet attrs, int defStyle) {
54          super(context, attrs, defStyle);
55          init(context);
56      }
57
58      private void init(Context context) {
59          setOrientation(HORIZONTAL);
60          LayoutInflater inflater = LayoutInflater.from(
                context);
61          inflater.inflate(R.layout.view_row_editor, this,
                true);
62
63          lineNumbers = (LineNumberTextView) findViewById(
                R.id.row_editor_line_numbers);
64          editText = (LinedEditorEditText) findViewById(R.
                id.row_editor_edit_text);
65          editText.addTextChangedListener(new TextWatcher
                () {
66              @Override
67              public void beforeTextChanged(CharSequence s
                    , int start, int count, int after) {}
68
69              @Override
70              public void onTextChanged(CharSequence s,
                    int start, int before, int count) {}
71
72              @Override
73              public void afterTextChanged(Editable s) {
74                  scrollToTextChange();
75              }
76          });
77
78          mScroller = new Scroller(context);
79          setFocusable(true);
80          setDescendantFocusability(
                FOCUS_AFTER_DESCENDANTS);
81          setWillNotDraw(false);
82          final ViewConfiguration config =
                ViewConfiguration.get(context);
83          mTouchSlop = config.getScaledTouchSlop();
84          mMinimumVelocity = config.
                getScaledMinimumFlingVelocity();
85          //???
86  //          editText.requestFocus();
87  //          editText.setSelection(editText.getText().
        length());
88      }
89
90      @Override
91      protected void onMeasure(int widthMeasureSpec, int
            heightMeasureSpec) {
92          super.onMeasure(widthMeasureSpec,
                heightMeasureSpec);
93          updateEditorLines();
94      }
95
96      public void updateEditorLines() {
97          mScroller.forceFinished(true);
98          int lineCount = editText.getLineCount();
99          lineNumbers.updateLineNumbers(lineCount);
100         editText.setMinWidth(getWidth() - lineNumbers.
                getWidth());
101     }
102
103     public String[] getPattern() {
104         String patternString = editText.getText().
                toString();
105         return PatternParser.parseRowFormatToPojo(
                patternString);
106     }
107
108     public void setPattern(String[] patternRows) {
109         final String pattern = PatternParser.
                parsePojoToRowFormat(patternRows);
110         //not updating textview when calling setText(
                pattern) - not running on UI thread?
111         post(new Runnable() {
112             @Override
113             public void run() {
114                 editText.setText(pattern);
115             }
116         });
117         updateEditorLines();
118     }
119
120     public EditText getEditText() {
121         return editText;
122     }
123
124     public void disableEditable() {
125         editText.setCursorVisible(false);
126         editText.setFocusableInTouchMode(false);
127         editText.setTextIsSelectable(false);
128         editText.clearFocus();
129     }
130
131     public void onEnterPressed() {
132         if(editText.getLineCount() + 1 > Constants.
                MAX_ROWS_AND_COLUMNS_LIMIT) {
133             Snackbar.make(this, getResources().getString
                    (R.string.info_max_rows, Constants.
                    MAX_ROWS_AND_COLUMNS_LIMIT), Snackbar.
                    LENGTH_SHORT).show();
134         } else {
135             editText.dispatchKeyEvent(new KeyEvent(
                    KeyEvent.ACTION_DOWN, KeyEvent.
                    KEYCODE_ENTER));
```

```
136             updateEditorLines();
137             scrollToTextChange();
138         }
139     }
140
141     public void onDeletePressed() {
142         editText.dispatchKeyEvent(new KeyEvent(KeyEvent.
                ACTION_DOWN, KeyEvent.KEYCODE_DEL));
143         updateEditorLines();
144     }
145
146     @Override
147     public boolean onTouchEvent(MotionEvent event) {
148         if (!canScroll()) return false;
149         if (mVelocityTracker == null) {
150             mVelocityTracker = VelocityTracker.obtain();
151         }
152         mVelocityTracker.addMovement(event);
153
154         final int action = event.getAction();
155         final float x = event.getX();
156         final float y = event.getY();
157         switch (action) {
158             case MotionEvent.ACTION_DOWN:
159                 //interrupt fling
160                 if (!mScroller.isFinished()) mScroller.
                        abortAnimation();
161                 mLastMotion.set(x, y);
162                 break;
163             case MotionEvent.ACTION_MOVE:
164                 int deltaX = (int) (mLastMotion.x - x);
165                 int deltaY = (int) (mLastMotion.y - y);
166                 mLastMotion.set(x, y);
167                 if (deltaX < 0) {
168                     if (getScrollX() < 0) {
169                         deltaX = 0;
170                     }
171                 } else if (deltaX > 0) {
172                     final int rightEdge = getWidth() -
                            getPaddingRight();
173                     final int availableToScroll =
                            getChildAt(1).getRight() -
                            getScrollX() - rightEdge;
174                     if (availableToScroll > 0) {
175                         deltaX = Math.min(
                                availableToScroll, deltaX);
176                     } else {
177                         deltaX = 0;
178                     }
179                 }
180                 if (deltaY < 0) {
181                     if (getScrollY() < 0) {
182                         deltaY = 0;
183                     }
184                 } else if (deltaY > 0) {
185                     final int bottomEdge = getHeight() -
                            getPaddingBottom();
186                     final int availableToScroll =
                            getChildAt(0).getBottom() -
                            getScrollY() - bottomEdge;
187                     if (availableToScroll > 0) {
188                         deltaY = Math.min(
                                availableToScroll, deltaY);
189                     } else {
190                         deltaY = 0;
191                     }
192                 }
193                 if (deltaY != 0 || deltaX != 0) {
194                     scrollBy(deltaX, deltaY);
195                 }
196                 break;
197             case MotionEvent.ACTION_UP:
198                 final VelocityTracker velocityTracker =
                        mVelocityTracker;
199                 velocityTracker.computeCurrentVelocity
                        (1000);
200                 int initialXVelocity = (int)
                        velocityTracker.getXVelocity();
201                 int initialYVelocity = (int)
                        velocityTracker.getYVelocity();
202
203                 if ((Math.abs(initialXVelocity) + Math.
                        abs(initialYVelocity) >
                        mMinimumVelocity) && getChildCount()
                        > 0) {
204                     fling(-initialXVelocity, -
                            initialYVelocity);
205                 }
206                 if (mVelocityTracker != null) {
207                     mVelocityTracker.recycle();
208                     mVelocityTracker = null;
209                 }
210                 break;
211         }
212         return true;
213     }
214
215     private boolean canScroll() {
216         int childCount = getChildCount();
217         if (childCount > 0) {
218             int childrenHeight = 0;
219             int childrenWidth = 0;
220             for (int i = 0; i < childCount; i++) {
221                 View child = getChildAt(i);
222                 childrenHeight += child.getHeight();
223                 childrenWidth += child.getWidth();
224             }
225             return (getHeight() < childrenHeight +
                    getPaddingTop() + getPaddingBottom())
226                     || (getWidth() < childrenWidth +
                            getPaddingRight() +
                            getPaddingLeft());
227         }
228         return false;
229     }
230
231     @Override
232     public boolean onInterceptTouchEvent(MotionEvent ev)
            {
233         final int action =ev.getAction();
234         if ((action == MotionEvent.ACTION_MOVE) &&
```

```
                mIsBeingDragged) {
235             return true;
236         }
237         if (!canScroll()) {
238             mIsBeingDragged = false;
239             return false;
240         }
241         final float x = ev.getX();
242         final float y = ev.getY();
243         switch (action) {
244             case MotionEvent.ACTION_MOVE:
245                 final int xDiff = (int) Math.abs(x -
                        mLastMotion.x);
246                 final int yDiff = (int) Math.abs(y -
                        mLastMotion.y);
247                 if (xDiff > mTouchSlop || yDiff >
                        mTouchSlop) {
248                     mIsBeingDragged = true;
249                 }
250                 break;
251             case MotionEvent.ACTION_DOWN:
252                 mLastMotion.x = x;
253                 mLastMotion.y = y;
254                 mIsBeingDragged = !mScroller.isFinished
                        ();
255                 break;
256             case MotionEvent.ACTION_UP:
257                 //release drag
258                 mIsBeingDragged = false;
259                 break;
260         }
261         //only intercept motion events if we are
                dragging
262         return mIsBeingDragged;
263     }
264
265     @Override
266     protected int computeVerticalScrollRange() {
267         return getChildCount() == 0 ? getHeight() :
                getChildrenDimensions().bottom;
268     }
269
270     @Override
271     protected int computeHorizontalScrollRange() {
272         return getChildCount() == 0 ? getWidth() :
                getChildrenDimensions().right;
273     }
274
275     @Override
276     protected void measureChild(View child, int
            parentWidthMeasureSpec, int
            parentHeightMeasureSpec) {
277         ViewGroup.LayoutParams lp = child.
                getLayoutParams();
278         int childWidthMeasureSpec;
279         int childHeightMeasureSpec;
280         childWidthMeasureSpec = getChildMeasureSpec(
                parentWidthMeasureSpec, getPaddingLeft()
                    + getPaddingRight(), lp.width);
281         childHeightMeasureSpec = MeasureSpec.
                makeMeasureSpec(0, MeasureSpec.UNSPECIFIED);
282         child.measure(childWidthMeasureSpec,
283                 childHeightMeasureSpec);
284     }
285
286     @Override
287     protected void measureChildWithMargins(View child,
            int parentWidthMeasureSpec, int widthUsed,
288                                            int
                                                    parentHeightMeasureSpec
                                                    , int
                                                    heightUsed
                                                    ) {
289         final MarginLayoutParams lp = (
                MarginLayoutParams) child.getLayoutParams();
290         final int childWidthMeasureSpec = MeasureSpec.
                makeMeasureSpec(lp.leftMargin + lp.
                rightMargin,
291                         MeasureSpec.UNSPECIFIED);
292         final int childHeightMeasureSpec = MeasureSpec.
                makeMeasureSpec(lp.topMargin + lp.
                bottomMargin,
293                         MeasureSpec.UNSPECIFIED);
294         child.measure(childWidthMeasureSpec,
                childHeightMeasureSpec);
295     }
296
297     @Override
298     public void computeScroll() {
299         if (mScroller.computeScrollOffset()) {
300             int oldX = getScrollX();
301             int oldY = getScrollY();
302             int x = mScroller.getCurrX();
303             int y = mScroller.getCurrY();
304             if (getChildCount() > 0) {
305                 Rect childrenDimens =
                        getChildrenDimensions();
306                 scrollTo(clamp(x, getWidth() -
                        getPaddingRight() - getPaddingLeft()
                        , childrenDimens.width()),
307                         clamp(y, getHeight() -
                                getPaddingBottom() -
                                getPaddingTop(),
                                childrenDimens.height()));
308             } else {
309                 scrollTo(x, y);
310             }
311             if (oldX != getScrollX() || oldY !=
                    getScrollY()) {
312                 onScrollChanged(getScrollX(), getScrollY
                        (), oldX, oldY);
313             }
314             // Keep on drawing until the animation has
                    finished.
315             postInvalidate();
316         }
317     }
318
319     private Rect getChildrenDimensions() {
320         int childCount = getChildCount();
321         if (childCount > 0) {
322             int width = 0;
323             int height = getChildAt(0).getHeight();
```

```
            for (int i = 0; i < childCount; i++) {
                View child = getChildAt(i);
                width += child.getWidth();
            }
            return new Rect(0, 0, width, height);
        }
        return null;
    }

    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        super.onLayout(changed, l, t, r, b);
        // initial clam
        scrollTo(getScrollX(), getScrollY());
    }

    public void fling(int velocityX, int velocityY) {
        if (getChildCount() > 0) {
            int height = getHeight() - getPaddingBottom() - getPaddingTop();
            int bottom = getChildAt(0).getHeight();
            int width = getWidth() - getPaddingRight() - getPaddingLeft();
            int right = getChildAt(1).getRight();
            mScroller.fling(getScrollX(), getScrollY(), velocityX, velocityY, 0, right - width, 0, bottom - height);
            invalidate();
        }
    }

    @Override
    public void scrollTo(int x, int y) {
        // we rely on the fact View.scrollBy calls scrollTo.
        if (getChildCount() > 0) {
            Rect childrenDimens = getChildrenDimensions();
            x = clamp(x, getWidth() - getPaddingRight() - getPaddingLeft(), childrenDimens.width());
            y = clamp(y, getHeight() - getPaddingBottom() - getPaddingTop(), childrenDimens.height());
            if (x != getScrollX() || y != getScrollY()) {

                mLastScrollTo.set(x, y);
                super.scrollTo(x, y);
            }
        }
    }

    private int clamp(int currentPos, int viewDimens, int childDimens) {
        if (viewDimens >= childDimens || currentPos < 0) {
            return 0;
        }
        if (viewDimens + currentPos > childDimens) {
            return childDimens - viewDimens;
        }
        return currentPos;
    }

    /**
     * scroll behavior so far: checking if point is in visible area works and scrolls to point if it is
     * not visible. edge behavior is faulty:
     * bottom: scrollTo() will clamp if line gets added at the bottom, will only scroll to second to last line,
     * since textview height has not been adjusted yet at that point.
     * right: {#LinedEditorEditText#getCursorPosition} gives wrong x position if character is added
     * on last word from {#@string/lorem_long_with_breaks}. I believe that is because a normal edittext
     * is made to wrap at the end of it's width. Since I am suppressing that behavior, and setting
     * the edittext's minimum width new after each interaction by calling {#updateEditorLines}, it is
     * likely that the implementation gets confused. The x position that is returned is always
     * where the added character would be if we wrapped the line.
     */
    public void scrollToTextChange() {
        //add line numbers width to get total width
        Point position = editText.getCursorPosition();
        position.x = position.x + lineNumbers.getWidth();
        Point center = getScreenCenter();
        int scrollX = getScrollX();
        int scrollY = getScrollY();
        boolean visible = new Rect(scrollX, scrollY, getWidth() + scrollX, getHeight() + scrollY)
            .contains(
                position.x,
                position.y
        );
        if (!visible) {
            int x = position.x - center.x;
            int y = position.y - center.y;
            scrollTo(x, y);
            invalidate();
        }
    }

    private Point getScreenCenter() {
        return new Point(getWidth() / 2, getHeight() / 2);
    }
}
```

Listing B.27: RowEditorLinearLayout.java

```java
package de.muffinworks.knittingapp;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.ImageButton;
import android.widget.TextView;

import java.io.IOException;

import de.muffinworks.knittingapp.layouts.RowEditorLinearLayout;
import de.muffinworks.knittingapp.storage.models.Pattern;
import de.muffinworks.knittingapp.util.Constants;
import de.muffinworks.knittingapp.views.PatternGridView;

public class ViewerActivity extends BaseActivity {

    private int mCurrentRow = 1;
    private TextView mRowText;
    private FrameLayout mPatternContainer;

    private PatternGridView mGridEditor;
    private RowEditorLinearLayout mRowEditor;
    private boolean mIsRowEditorActive = false;

    private Pattern mPattern;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_viewer);

        enableBackInActionBar(true);

        initEditors();

        String patternId = getIntent().getStringExtra(Constants.EXTRA_PATTERN_ID);
        if (patternId != null) {
            mPattern = mStorage.load(patternId);
            mRowEditor.setPattern(mPattern.getPatternRows());
            setActionBarTitle(mPattern.getName());
        }
        initCounter();
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.reset_row_counter) {
            updateRowCounter(1);
            mGridEditor.scrollCurrentRowToCenter();
            return true;
        } else if (id == R.id.switch_view_style) {
            switchEditors();
            return true;
        } else if (id == R.id.scroll_current_row_to_center) {
            mGridEditor.scrollCurrentRowToCenter();
        } else if (id == R.id.open_editor) {
            Intent intent = new Intent(this, EditorActivity.class);
            intent.putExtra(Constants.EXTRA_PATTERN_ID, mPattern.getId());
            startActivityForResult(intent, Constants.REQUEST_CODE_EDITOR);
        } else if (id == R.id.open_glossary) {
            Intent intent = new Intent(this, GlossaryActivity.class);
            startActivity(intent);
        } else if (id == R.id.export_pattern) {
            exportPattern();
        }
        return super.onOptionsItemSelected(item);
    }

    private void exportPattern() {
        try {
            mStorage.export(mPattern.getId());
            showAlertDialog(getString(R.string.success_export_pattern, Constants.EXPORT_DIR));
        } catch (IOException e) {
            showAlertDialog(getString(R.string.error_export));
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_viewer, menu);
        return true;
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == Constants.REQUEST_CODE_EDITOR) {
            if (resultCode == Activity.RESULT_OK) {
                //user changed pattern and saved -> viewer needs to refresh data
                mPattern = mStorage.load(mPattern.getId());
                mRowEditor.setPattern(mPattern.getPatternRows());
                if (mGridEditor != null) {
                    mGridEditor.setPattern(mPattern.getPatternRows());
                }
                setActionBarTitle(mPattern.getName());
            } else if (resultCode == Activity.RESULT_CANCELED) {
                if (data != null) {
                    boolean wasPattenDeleted = data.
```

```
                        getBooleanExtra(Constants.
                        EXTRA_PATTERN_DELETED, false);
104                     if (wasPattenDeleted) {
105                         finish();
106                     }
107                 }
108             }
109         }
110     }
111
112     private void initCounter() {
113         mRowText = (TextView) findViewById(R.id.row);
114         updateRowCounter();
115
116         ImageButton mIncreaseRow = (ImageButton)
                findViewById(R.id.button_increase);
117         mIncreaseRow.setOnClickListener(new View.
                OnClickListener() {
118             @Override
119             public void onClick(View v) {
120                 updateRowCounter(mCurrentRow +1);
121             }
122         });
123
124         ImageButton mDecreaseRow = (ImageButton)
                findViewById(R.id.button_decrease);
125         mDecreaseRow.setOnClickListener(new View.
                OnClickListener() {
126             @Override
127             public void onClick(View v) {
128                 updateRowCounter(mCurrentRow -1);
129             }
130         });
131     }
132
133     private void updateRowCounter(int rows) {
134         int maxRows = mPattern == null ? Constants.
                DEFAULT_ROWS : mPattern.getRows();
135         mCurrentRow = Math.min(Math.max(rows, 1),
                maxRows);
136         mRowText.setText(Integer.toString(mCurrentRow));
137         if (mPattern != null) {
138             mPattern.setCurrentRow(mCurrentRow);
139             mStorage.save(mPattern);
140         }
141         if (mGridEditor != null) {
142             mGridEditor.setCurrentRow(mCurrentRow);
143         }
144     }
145
146     private void updateRowCounter() {
147         if (mPattern != null) {
148             mCurrentRow = mPattern.getCurrentRow();
149         }
150         updateRowCounter(mCurrentRow);
151     }
152
153     private void initEditors() {
154         mPatternContainer = (FrameLayout) findViewById(R
                .id.editor_container);
155
156         mGridEditor = new PatternGridView(this);
157         mGridEditor.setCanBeEdited(false);
158
159         mRowEditor = new RowEditorLinearLayout(this);
160         mRowEditor.disableEditable();
161
162         mPatternContainer.addView(mGridEditor);
163     }
164
165     private void switchEditors() {
166         if (!mIsRowEditorActive) {
167             mPatternContainer.removeAllViews();
168             mPatternContainer.addView(mRowEditor);
169         } else {
170             mPatternContainer.removeAllViews();
171             mPatternContainer.addView(mGridEditor);
172             mGridEditor.setPattern(mPattern.
                    getPatternRows());
173         }
174         mIsRowEditorActive = !mIsRowEditorActive;
175     }
176 }
```

Listing B.28: ViewerActivity.java

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <android.support.design.widget.CoordinatorLayout
3       xmlns:android="http://schemas.android.com/apk/res/
            android"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent">
6
7       <FrameLayout
8           android:id="@+id/fragment_container"
9           android:layout_width="match_parent"
10          android:layout_height="match_parent"/>
11
12  </android.support.design.widget.CoordinatorLayout>
```

Listing B.29: activity_editor.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/glossary_listview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:divider="@null"/>

</LinearLayout>
```

Listing B.30: activity_glossary.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    android:id="@+id/coord"
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="de.muffinworks.knittingapp.
        PatternListActivity">

    <ListView
        android:id="@+id/patterns_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="100dp"
        android:clipToPadding="false"/>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@drawable/ic_add_white_24dp" />

</android.support.design.widget.CoordinatorLayout>
```

Listing B.31: activity_pattern_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="11">

    <FrameLayout
        android:id="@+id/editor_container"
        android:textSize="80sp"
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="8"/>

    <FrameLayout
        android:background="@color/colorPrimary"
        android:layout_width="match_parent"
        android:layout_height="3dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:orientation="horizontal"
        android:layout_weight="3">

        <LinearLayout
            android:id="@+id/containerRows"
            android:layout_gravity="left|center_vertical"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:background="@color/colorPrimary"
            android:layout_weight="1"
            android:gravity="center"
            android:orientation="vertical">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="top|
                    center_horizontal"
                android:textColor="@color/offblack"
                android:textSize="20sp"
                android:text="Reihe"/>

            <TextView
                android:id="@+id/row"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom|
                    center_horizontal"
                android:gravity="center"
                android:textColor="@color/offblack"
                android:textSize="70sp"
                android:text="120"/>
        </LinearLayout>

        <LinearLayout
            android:id="@+id/containerControls"
            android:orientation="horizontal"
            android:gravity="center"
            android:layout_weight="3"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:background="@color/colorAccent">
```

```
            <ImageButton
                android:id="@+id/button_increase"
                android:elevation="4dp"
                android:layout_width="150dp"
                android:layout_height="150dp"
                android:background="@drawable/ripple_round"
                android:src="@drawable/ic_add_white_24dp"/>

            <ImageButton
                android:id="@+id/button_decrease"
                android:layout_marginLeft="75dp"
                android:elevation="4dp"
                android:layout_width="75dp"
                android:layout_height="75dp"
                android:background="@drawable/ripple_round"
                android:src="@drawable/ic_remove_white_24dp"
                    />

        </LinearLayout>

    </LinearLayout>

</LinearLayout>
```

Listing B.32: activity_viewer.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>


</resources>
```

Listing B.33: attr.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#FFC107</color>
    <color name="colorPrimaryDark">#FFA000</color>
    <color name="colorPrimaryLight">#FFECB3</color>

    <color name="colorAccent">#607D8B</color>
    <color name="colorAccentDark">#4e6671</color>

    <color name="highlight_current_row">#95ffc107</color>

    <!-- Text color for a button in a pad. -->
    <color name="keyboard_button_text_color">#FFF</color>
    <!-- Ripple color when a button is pressed in a pad. -->
    <color name="keyboard_button_ripple_color">#1A000000</
        color>

    <color name="icons">#212121</color>
    <color name="primaryText">#212121</color>
    <color name="secondaryText">#727272</color>
    <color name="divider">#B6B6B6</color>

    <color name="offWhite">#dedede</color>
    <color name="black_01">#33dedede</color>
    <color name="black_10">#33a1a1a1</color>
    <color name="black_20">#33575757</color>
    <color name="black_30">#33292929</color>
    <color name="black_40">#66575757</color>
    <color name="black_60">#99575757</color>
    <color name="black_80">#cc575757</color>
    <color name="offblack">#2f2f2f</color>

    <color name="red_400">#EF5350</color>
</resources>
```

Listing B.34: colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:orientation="vertical"
    android:gravity="center"
    android:weightSum="2"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="center"
        android:layout_marginBottom="30dp"
        android:layout_weight="1">

        <TextView
            android:textSize="20sp"
            android:layout_marginRight="15dp"
            android:text="@string/colums"
            android:gravity="right"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

        <EditText
            android:id="@+id/edittext_columns"
            android:imeOptions="actionNext"
            android:inputType="number"
            android:digits="1234567890"
            android:layout_width="100dp"
            android:layout_height="50dp"/>

    </LinearLayout>
```

```
35
36      <LinearLayout
37          android:orientation="horizontal"
38          android:layout_width="match_parent"
39          android:layout_height="0dp"
40          android:gravity="center"
41          android:layout_weight="1">
42
43          <TextView
44              android:textSize="20sp"
45              android:layout_marginRight="15dp"
46              android:text="@string/rows"
47              android:gravity="right"
48              android:layout_width="wrap_content"
49              android:layout_height="wrap_content"/>
```

```
1   <resources>
2       <dimen name="fab_margin">16dp</dimen>
3       <dimen name="row_editor_default_text_size">50sp</dimen>
```

```
1   <resources>
2       <dimen name="activity_horizontal_margin">64dp</dimen>
3   </resources>
```

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout
3       xmlns:android="http://schemas.android.com/apk/res/
            android"
4       android:orientation="vertical"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       android:weightSum="4">
8
9       <de.muffinworks.knittingapp.views.PatternGridView
10          android:id="@+id/grid"
11          android:layout_width="match_parent"
12          android:layout_height="0dp"
13          android:layout_weight="3"/>
14
15      <FrameLayout
16          android:layout_width="match_parent"
17          android:layout_height="3dp"
18          android:background="@color/colorPrimary"/>
19
20      <LinearLayout
21          android:orientation="horizontal"
22          android:background="@color/colorPrimary"
23          android:layout_width="match_parent"
24          android:layout_height="0dp"
25          android:layout_weight="1"
26          android:weightSum="7">
27
28          <GridView
29              android:id="@+id/keyboard_gridview"
30              android:background="@color/colorAccent"
```

```
50
51          <EditText
52              android:id="@+id/edittext_rows"
53              android:imeOptions="actionDone"
54              android:inputType="number"
55              android:digits="1234567890"
56              android:layout_width="100dp"
57              android:layout_height="50dp"/>
58
59      </LinearLayout>
60  </LinearLayout>
```

Listing B.35: dialog_set_grid_size.xml

```
4   </resources>
```

Listing B.36: dimens.xml

Listing B.37: dimens-w820.xml

```
31              android:fadeScrollbars="false"
32              android:layout_width="0dp"
33              android:layout_weight="6"
34              android:scrollbarSize="10dp"
35              android:layout_height="wrap_content"
36              android:horizontalSpacing="20dp"
37              android:numColumns="4"/>
38
39          <LinearLayout
40              android:id="@+id/grid_delete_button_container"
41              android:background="@color/colorPrimary"
42              android:layout_height="match_parent"
43              android:layout_width="0dp"
44              android:layout_weight="1">
45
46              <ImageButton
47                  android:src="@drawable/ic_delete_white_48dp"
48                  android:layout_width="match_parent"
49                  android:layout_height="match_parent"
50                  android:background="@drawable/
                        keyboard_button_background"
51                  android:onClick="onDeleteToggled" />
52
53          </LinearLayout>
54
55      </LinearLayout>
56
57  </LinearLayout>
```

Listing B.38: fragment_editor_grid.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="3">

    <de.muffinworks.knittingapp.layouts.
        RowEditorLinearLayout
        android:id="@+id/row_editor_container"
        android:layout_weight="2"
        android:layout_width="match_parent"
        android:layout_height="0dp"/>

    <FrameLayout
        android:background="@color/colorPrimary"
        android:layout_width="match_parent"
        android:layout_height="3dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:weightSum="7"
        android:layout_weight="1">

        <GridView
            android:id="@+id/keyboard_gridview"
            android:background="@color/colorAccent"
            android:scrollbarSize="10dp"
            android:fadeScrollbars="false"
            android:horizontalSpacing="20dp"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="3"
            android:numColumns="3"/>

        <FrameLayout
            android:background="@color/colorAccentDark"
            android:layout_width="3dp"
            android:layout_height="match_parent"/>

        <include
            layout="@layout/view_numpad"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="3"/>

        <LinearLayout
            android:orientation="vertical"
            android:background="@color/colorPrimary"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:weightSum="2"
            android:layout_weight="1">

            <ImageButton
                android:layout_margin="0dp"
                android:layout_height="0dp"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:src="@drawable/
                    ic_backspace_white_24dp"
                android:id="@+id/op_delete"
                style="@style/FontButtonStyle.KeyEvent"
                android:contentDescription="delete"
                android:onClick="onDelete" />

            <ImageButton
                android:layout_margin="0dp"
                android:layout_height="0dp"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:src="@drawable/
                    ic_keyboard_return_white_24dp"
                android:id="@+id/op_enter"
                style="@style/FontButtonStyle.KeyEvent"
                android:contentDescription="enter"
                android:onClick="onEnter" />

        </LinearLayout>

    </LinearLayout>

</LinearLayout>
```

Listing B.39: fragment_editor_row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="de.muffinworks.knittingapp.EditorActivity
        ">

    <item
        android:id="@+id/export_pattern"
        android:orderInCategory="100"
        android:title="@string/menu_export_pattern"
        app:showAsAction="never" />

    <item
        android:id="@+id/set_size"
        android:orderInCategory="100"
        android:title="@string/menu_grid_size"
        app:showAsAction="ifRoom" />

    <item
        android:id="@+id/open_glossary"
        android:orderInCategory="100"
        android:title="@string/menu_open_glossary"
        app:showAsAction="never" />

    <item
        android:id="@+id/switch_editor"
        android:orderInCategory="100"
        android:icon="@drawable/ic_swap_horiz_black_24dp"
        android:title="@string/menu_switch_editor"
        app:showAsAction="always" />

    <item
        android:id="@+id/save_pattern"
```

```
        android:orderInCategory="100"
        android:icon="@drawable/ic_save_black_24dp"
        android:title="@string/menu_save"
        app:showAsAction="always"/>

    <item
        android:id="@+id/edit_pattern_name"
        android:orderInCategory="100"
        android:title="@string/menu_edit_name"
        app:showAsAction="never" />

    <item
        android:id="@+id/delete_pattern"
        android:orderInCategory="100"
        android:title="@string/menu_delete_pattern"
        app:showAsAction="never" />

</menu>
```

Listing B.40: menu_editor.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/export_all"
        android:orderInCategory="100"
        android:title="@string/menu_export_all"
        app:showAsAction="never" />

    <item
        android:id="@+id/import_pattern"
        android:orderInCategory="100"
        android:title="@string/menu_import_pattern"
        app:showAsAction="never" />

</menu>
```

Listing B.41: menu_editor_pattern_list.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/
    android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="de.muffinworks.knittingapp.MainActivity">

    <item
        android:id="@+id/export_pattern"
        android:orderInCategory="100"
        android:title="@string/menu_export_pattern"
        app:showAsAction="never" />

    <item
        android:id="@+id/open_glossary"
        android:orderInCategory="100"
        android:title="@string/menu_open_glossary"
        app:showAsAction="never" />

    <item
        android:id="@+id/reset_row_counter"
        android:orderInCategory="100"
        android:title="@string/menu_reset_counter"
        app:showAsAction="never" />

    <item
        android:id="@+id/open_editor"
        android:orderInCategory="100"
        android:title="@string/menu_open_editor"
        app:showAsAction="never" />

    <item
        android:id="@+id/scroll_current_row_to_center"
        android:orderInCategory="100"
        android:title="@string/menu_jump_to_current_row"
        app:showAsAction="always" />

    <item
        android:id="@+id/switch_view_style"
        android:icon="@drawable/ic_swap_horiz_black_24dp"
        android:orderInCategory="100"
        android:title="@string/menu_switch_editor"
        app:showAsAction="always" />
</menu>
```

Listing B.42: menu_viewer.xml

```
<resources>
    <string name="app_name">Knitting App</string>

    <!--dialog related-->
    <string name="dialog_ok">OK</string>
    <string name="dialog_yes">Yes</string>
    <string name="dialog_no">No</string>
    <string name="dialog_cancel">Cancel</string>
    <string name="dialog_title_grid_size">Change grid size</
        string>
    <string name="dialog_title_pattern_delete">Delete
        pattern %1$s?</string>
    <string name="dialog_title_pattern_name">Pattern name</
        string>
    <string name="dialog_title_pattern_save_changes">Save
        changes?</string>
```

```
    <!--Activity titles-->
    <string name="activity_title_glossary">Glossar</string>

    <!--errors-->
    <string name="error_over_max_size">Only %1$s supported</
        string>
    <string name="error_must_implement_interface"
        translatable="false"> must implement %1$s</string>
    <string name="error_load_metadata">Could not close file
        reader</string>
    <string name="error_update_metadata">Could not write
        metadata</string>
    <string name="error_save_pattern">Could not write
        pattern to disk</string>
    <string name="error_file_not_found">Could not find file:
         %1$s</string>
    <string name="error_dimension_zero">Must be at least 1</
        string>
    <string name="error_external_storage_not_writable">
        External storage is not writable</string>
    <string name="error_external_storage_not_mounted">
        External storage not available</string>
    <string name="error_export">Export failed</string>
    <string name="error_import_no_json">Import failed: can\'
        t read file</string>

    <!--success-->
    <string name="success_export_pattern">Successfully
        exported pattern to folder %1$s on SD card</string>
    <string name="success_export_all">Successfully exported
        to folder %1$s on SD card</string>
    <string name="success_save_pattern">Saving successful</
        string>

    <!--info-->
    <string name="info_import_pattern_already_exists">This
        pattern already exists. Do you want to overwrite the
         existing file?</string>
    <string name="info_storage_permission">Allow access to
        storage to export and import patterns</string>
    <string name="info_max_rows">Only %1$d rows are
        supported</string>

    <!--fragment tags-->
    <string name="tag_dialog_fragment_grid_size"
        translatable="false">dialog_fragment_grid_size</
        string>
    <string name="tag_dialog_fragment_edit_name"
        translatable="false">dialog_fragment_edit_name</
        string>
    <string name="tag_dialog_fragment_delete_pattern"
        translatable="false">dialog_fragment_delete_pattern<
        /string>
    <string name="tag_dialog_fragment_set_name" translatable
        ="false">dialog_fragment_set_name</string>

    <!--menu-->
    <string name="menu_grid_size">Grid size</string>
    <string name="menu_switch_editor">Switch view</string>
    <string name="menu_save">save</string>
    <string name="menu_edit_name">Change pattern name</
        string>
    <string name="menu_delete_pattern">Delete pattern</
        string>
    <string name="menu_reset_counter">Reset counter</string>
    <string name="menu_open_glossary">Open glossar</string>
    <string name="menu_open_editor">Edit pattern</string>
    <string name="menu_jump_to_current_row">Current row</
        string>
    <string name="menu_export_all">Export all</string>
    <string name="menu_export_pattern">Export pattern</
        string>
    <string name="menu_import_pattern">Import pattern</
        string>

    <string name="rows">rows</string>
    <string name="colums">columns</string>
    <string name="placeholder_pattern_name">Pattern name</
        string>
    <string name="placeholder_line_numbers" translatable="
        false">1\n2\n3\n4\n5\n6</string>
</resources>
```

Listing B.43: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Knitting App</string>
    <string name="activity_title_glossary">Glossar</string>
    <string name="colums">Spalten</string>
    <string name="dialog_cancel">Abbrechen</string>
    <string name="dialog_no">Nein</string>
    <string name="dialog_ok">OK</string>
    <string name="dialog_title_grid_size">Gittergröße ändern
        </string>
    <string name="dialog_title_pattern_delete">Strickmuster
        %1$s wirklich löschen?</string>
    <string name="dialog_title_pattern_name">
        Strickmustername</string>
    <string name="dialog_title_pattern_save_changes">Ä
        nderungen speichern?</string>
    <string name="dialog_yes">Ja</string>
    <string name="error_dimension_zero">Muss mindestens 1
        sein</string>
    <string name="error_export">Fehler beim exportieren</
        string>
    <string name="error_external_storage_not_mounted">
        Externer Speicher nicht verfügbar!</string>
    <string name="error_external_storage_not_writable">
        Externer Speicher kann nicht beschrieben werden!</
        string>
    <string name="error_file_not_found">Datei: %1$s konnte
        nicht gefunden werden!</string>
    <string name="error_load_metadata">File reader konnte
        nicht geschlossen werden</string>
    <string name="error_must_implement_interface"></string>
    <string name="error_over_max_size">Nur %1$s unterstützt<
        /string>
    <string name="error_save_pattern">Muster konnte nicht
        gespeichert werden</string>
    <string name="error_update_metadata">Metadata konnte
        nicht gespeichert werden</string>
    <string name="info_import_pattern_already_exists">Dieses
         Muster exisistiert bereits. Muster überschreiben?</
```

```
        string>
25      <string name="info_max_rows">Nur maximal %1$d Reihen
            unterstützt</string>
26      <string name="info_storage_permission">Zugriff erlauben
            zum Exportieren und Importieren von Mustern</string>
27      <string name="menu_delete_pattern">Muster löschen</
            string>
28      <string name="menu_edit_name">Musternamen ändern</string
            >
29      <string name="menu_export_all">Alle exportieren</string>
30      <string name="menu_export_pattern">Muster exportieren</
            string>
31      <string name="menu_grid_size">Mustergröße</string>
32      <string name="menu_import_pattern">Muster importieren</
            string>
33      <string name="menu_jump_to_current_row">Aktuelle Reihe</
            string>
34      <string name="menu_open_editor">Muster bearbeiten</
            string>
35      <string name="menu_open_glossary">Glossar öffnen</string
            >
36      <string name="menu_reset_counter">Zähler zurücksetzen</
        string>
37      <string name="menu_save">Speichern</string>
38      <string name="menu_switch_editor">Ansicht wechseln</
            string>
39      <string name="placeholder_line_numbers" translatable="
            false"></string>
40      <string name="placeholder_pattern_name">Mustername</
            string>
41      <string name="rows">Reihen</string>
42      <string name="success_export_all">Muster erfolgreich
            nach %1$s auf SD Karte exportiert</string>
43      <string name="success_save_pattern">Speichern
            erfolgreich</string>
44      <string name="success_export_pattern">Muster erfolgreich
             nach %1$s auf SD Karte exportiert</string>
45      <string name="error_import_no_json">Import
            fehlgeschlagen: Datei kann nicht gelesen werden. </
            string>
46  </resources>
```

Listing B.44: strings-de.xml

```
1   <resources>
2
3       <style name="AppTheme" parent="Theme.AppCompat.Light">
4           <item name="colorPrimary">@color/colorPrimary</item>
5           <item name="colorPrimaryDark">@color/
                colorPrimaryDark</item>
6           <item name="colorAccent">@color/colorAccent</item>
7           <!--https://stackoverflow.com/questions/26450936/
                actionmode-with-toolbar-offsetting-content-->
8           <item name="windowActionModeOverlay">true</item>
9       </style>
10
11      <style name="AppTheme.NoActionBar">
12          <item name="windowActionBar">false</item>
13          <item name="windowNoTitle">true</item>
14      </style>
15
16      <style name="AppTheme.AppBarOverlay" parent="
            ThemeOverlay.AppCompat.Dark.ActionBar" />
17
18      <style name="AppTheme.PopupOverlay" parent="ThemeOverlay
            .AppCompat.Light" />
19
20
21
22      <style name="RowTextStyle" parent="@style/Widget.
            AppCompat.EditText">
23          <item name="android:background">@android:color/
                transparent</item>
24          <item name="android:cursorVisible">false</item>
25          <item name="android:fontFamily">sans-serif-light</
                item>
26          <item name="android:includeFontPadding">false</item>
27          <item name="android:gravity">bottom|end</item>
28      </style>
29
30      <style name="FontButtonStyle" parent="@android:style/
            Widget.Material.Light.Button.Borderless">
31          <item name="android:layout_width">wrap_content</item
                >
32          <item name="android:layout_height">wrap_content</
                item>
33          <item name="android:background">@drawable/
                keyboard_button_background</item>
34          <item name="android:fontFamily">sans-serif-light</
                item>
35          <item name="android:gravity">center</item>
36          <item name="android:includeFontPadding">false</item>
37          <item name="android:minWidth">0dip</item>
38          <item name="android:minHeight">0dip</item>
39          <item name="android:textAllCaps">false</item>
40          <item name="android:textColor">@color/
                keyboard_button_text_color</item>
41      </style>
42
43      <style name="NumpadLayoutStyle">
44          <item name="android:layout_height">match_parent</
                item>
45          <item name="android:layout_width">0dip</item>
46          <item name="android:layout_weight">264</item>
47          <item name="android:paddingTop">12dip</item>
48          <item name="android:paddingBottom">20dip</item>
49          <item name="android:paddingLeft">12dip</item>
50          <item name="android:paddingStart">12dip</item>
51          <item name="android:paddingRight">12dip</item>
52          <item name="android:paddingEnd">12dip</item>
53      </style>
54
55  </resources>
```

Listing B.45: styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res
    /android">

    <style name="FontButtonStyle.Key">
        <item name="android:layout_margin">4dip</item>
        <item name="android:textSize">32sp</item>
    </style>

    <style name="FontButtonStyle.KeyEvent">
        <item name="android:layout_margin">8dip</item>
        <item name="android:textSize">23sp</item>
    </style>
</resources>
```

Listing B.46: styles-port.xml

```
<resources>
    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
        <item name="android:windowDrawsSystemBarBackgrounds"
            >true</item>
        <item name="android:statusBarColor">@android:color/
            transparent</item>
    </style>
</resources>
```

Listing B.47: styles-values-v21.xml

```
<?xml version="1.0" encoding="utf-8"?>
<de.muffinworks.knittingapp.views.KnittingFontButton
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    style="@style/FontButtonStyle.Key"
    android:padding="30dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</de.muffinworks.knittingapp.views.KnittingFontButton>
```

Listing B.48: view_grid_key.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/symbol"
        android:layout_width="75dp"
        android:layout_height="match_parent"
        android:layout_margin="16dp"
        android:textSize="32sp"
        android:text="k"
        android:gravity="center"/>

    <TextView
        android:id="@+id/symbol_description"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp"
        android:gravity="left|center"
        android:textSize="26sp"/>
</LinearLayout>
```

Listing B.49: view_item_glossary.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="75dp">

    <TextView
        android:gravity="center_vertical"
        android:textSize="25sp"
        android:layout_gravity="start|center_vertical"
        android:padding="15dp"
        android:id="@+id/pattern_name"
        android:text="@string/placeholder_pattern_name"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <ImageButton
        android:tint="@color/colorAccent"
        android:background="@drawable/
            keyboard_button_background"
        android:id="@+id/button_edit"
```

```
        android:src="@drawable/ic_mode_edit_black_24dp"
        android:layout_gravity="end|center_vertical"
        android:layout_width="50dp"
        android:layout_height="match_parent" />

    <ImageButton
        android:tint="@color/colorAccent"
        android:background="@drawable/
            keyboard_button_background"
        android:id="@+id/button_delete"
        android:src="@drawable/ic_delete_black_24dp"
        android:layout_gravity="end|center_vertical"
        android:layout_width="50dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

Listing B.50: view_item_list_pattern.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.android.calculator2.CalculatorPadLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    style="@style/NumpadLayoutStyle"
    android:rowCount="4"
    android:columnCount="3"
    android:background="@color/colorAccent">

    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <de.muffinworks.knittingapp.views.KnittingFontButton
        style="@style/FontButtonStyle.Key"
        android:onClick="onNumPadClick"
        android:text="0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</com.android.calculator2.CalculatorPadLayout>
```

Listing B.51: view_numpad.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/input"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:singleLine="true"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"/>

</LinearLayout>
```

Listing B.52: view_pattern_name_input.xml

```
<?xml version="1.0" encoding="utf-8"?>
<merge
    xmlns:android="http://schemas.android.com/apk/res/
        android">

    <de.muffinworks.knittingapp.views.LineNumberTextView
        android:cursorVisible="false"
        style="@style/RowTextStyle"
        android:elevation="4dp"
        android:id="@+id/row_editor_line_numbers"
        android:textSize="@dimen/
            row_editor_default_text_size"
        android:layout_width="10dp"
        android:layout_height="wrap_content"
        android:background="@color/black_20"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:text="@string/placeholder_line_numbers"/>

    <de.muffinworks.knittingapp.views.LinedEditorEditText
        android:paddingRight="50dp"
        style="@style/RowTextStyle"
        android:id="@+id/row_editor_edit_text"
        android:gravity="center_vertical|left"
        android:textSize="@dimen/
            row_editor_default_text_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</merge>
```

Listing B.53: view_row_editor.xml

**Eidesstattliche Erklärung**

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach anderen gedruckten oder im Internet verfügbaren Werken entnommen sind, habe ich durch genaue Quellenangaben kenntlich gemacht.

.....

(Ort, Datum, Unterschrift)