

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)

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Debora Weber-Wulff for the continuous support and guidance given to me during this thesis.

Besides my supervisor, I would also like to thank my proof-readers, Tu Le-Than, Tormod Gjeitnes Hellen, and Joakim Uddholm, for their hard work.

Last but not least, I would like to thank my family and friends for cheering me on and supporting me at all times.

Contents

Contents	i
1 Introduction	1
2 Background	3
2.1 Definition of Knitting Pattern and Knitting Pattern Chart	3
2.2 Comparison of Existing Solutions	4
2.2.1 Android apps	4
knit tink — Row Counter by Jennifer K. Warren	4
Knitting Counter by mkacki	5
Knitting and Crochet Buddy by Colorwork Apps	6
BeeCount knitting Counter by knirrr	7
Knitting Chart Maker by Awesome Applications	8
2.2.2 Other	9
KnitML by Jonathan Whitall	9
2.3 UI Evaluation	10
3 Requirements	11
3.1 Functional Requirements	11
3.2 Non-functional Requirements	13
4 Design	14
5 Android Basics	17
5.1 The Operating System Android	17
5.2 Basic Components of an Android App	17
5.2.1 Activity	17

5.2.2 Actionbar	19
5.2.3 Fragment	19
5.2.4 View	21
5.2.5 Storage	22
6 Implementation	23
6.1 Stitch symbols	23
6.2 Pattern and Parsing between Pattern Formats	24
6.3 Persistent Disk Storage	25
6.4 Displaying a Pattern	26
6.4.1 Grid Format	26
6.4.2 Row Format	29
6.5 Keyboard	31
6.6 Viewer with Row Counter	32
6.7 Editor	33
Editor Fragments	34
6.8 Pattern List	34
6.9 Glossary	35
7 User Test	38
8 Evaluation and Discussion	41
9 Outlook	43
Abbreviations	44
List of Figures	45
Listings	48
Bibliography	49
A User Interviews	i
A.1 Question catalogue	ii
A.2 Interview with Thilo Ilg	ii
A.3 Interview with Nadine Kost	iii
A.4 Interview with Angela Thomas	iv

CONTENTS

iii

B Source Code

vi

Chapter 1

Introduction

Since the beginning of the 2000s, knitting has encountered a steady rise in popularity the online article “Pride in the wool: the rise of knitting” written by Lewis 2011. This, she says, might be due to the rise of the internet and social media, and because of the increasingly important role they play in the daily life. The older knitting generation is adapting to new technology and switching over, Lewis explains, bringing knitting as a craft and hobby closer to the younger generations (Lewis 2011). Online communities like Ravelry¹ and Youtube² 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. As of August 2016 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/>

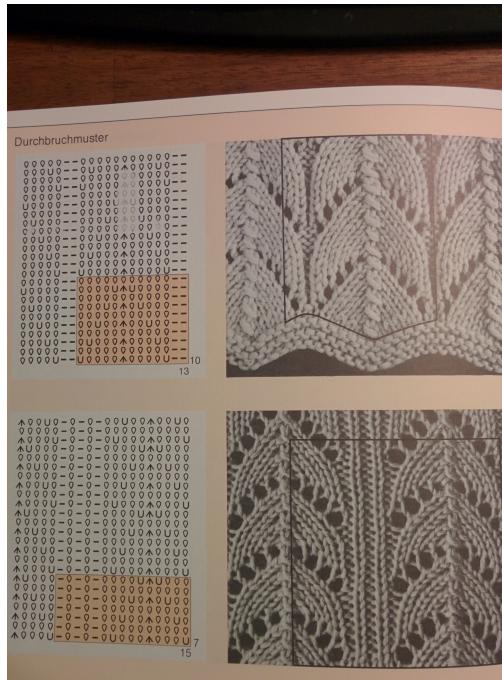


Figure 1.1: Knitting patterns and their corresponding pattern charts from Natter 1983, p142

Displaying a pattern chart on a mobile device is difficult because of the small size of the screens of contemporary mobile devices. 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, be executed iteratively throughout development to ensure a useful³ User Interface (UI) design.

³for the definition of “useful” see Section 2.3

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 generally limited in its ability to stretch and be formed.

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 finite 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 Japanese knitting pattern charts being published without a glossary 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.

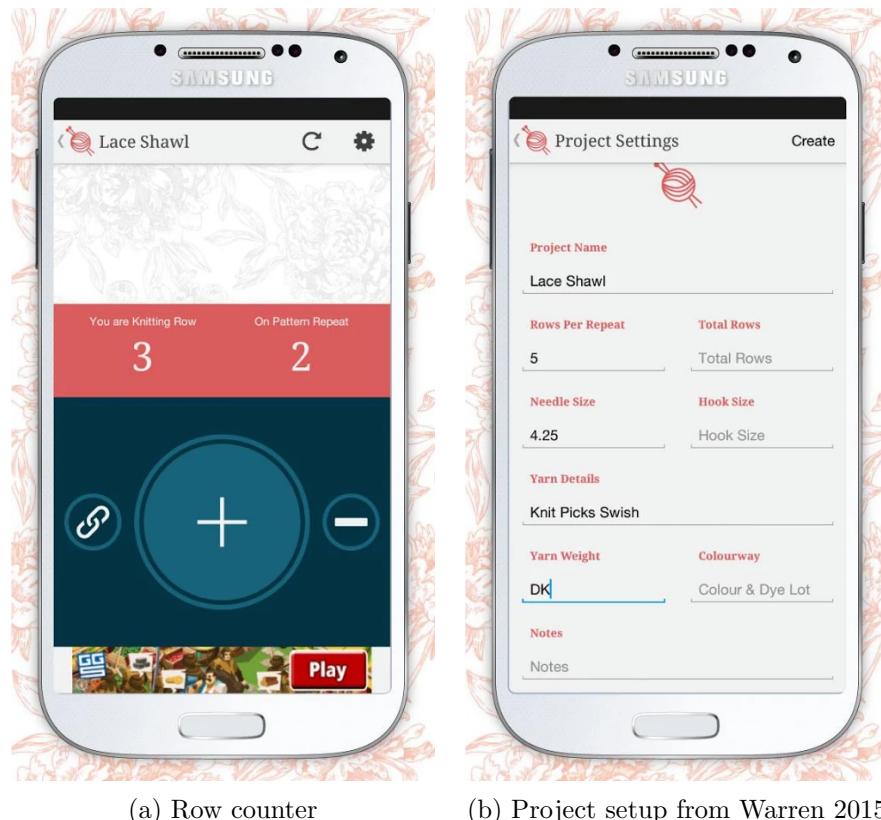
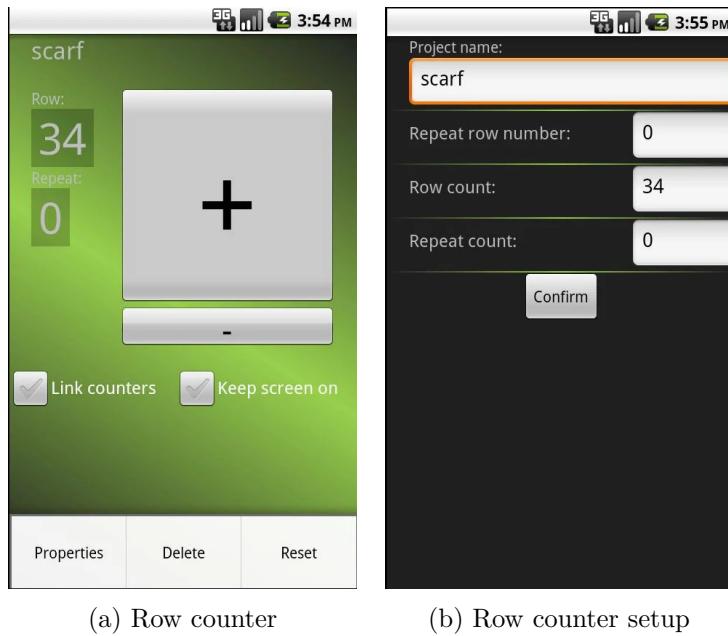


Figure 2.1: Screenshots of the app knit tink

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=androididdeveloperjoe.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: yarn and crochet symbol charts, an abbreviation chart, size charts for knitting needles and crocheting hooks, a project timer, a ruler function, and a flashlight.

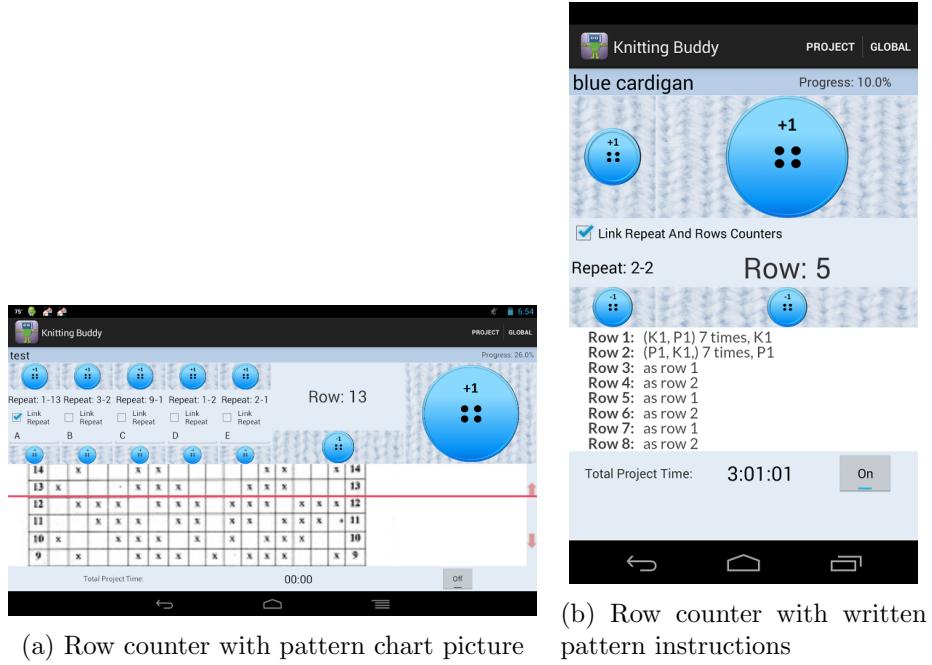
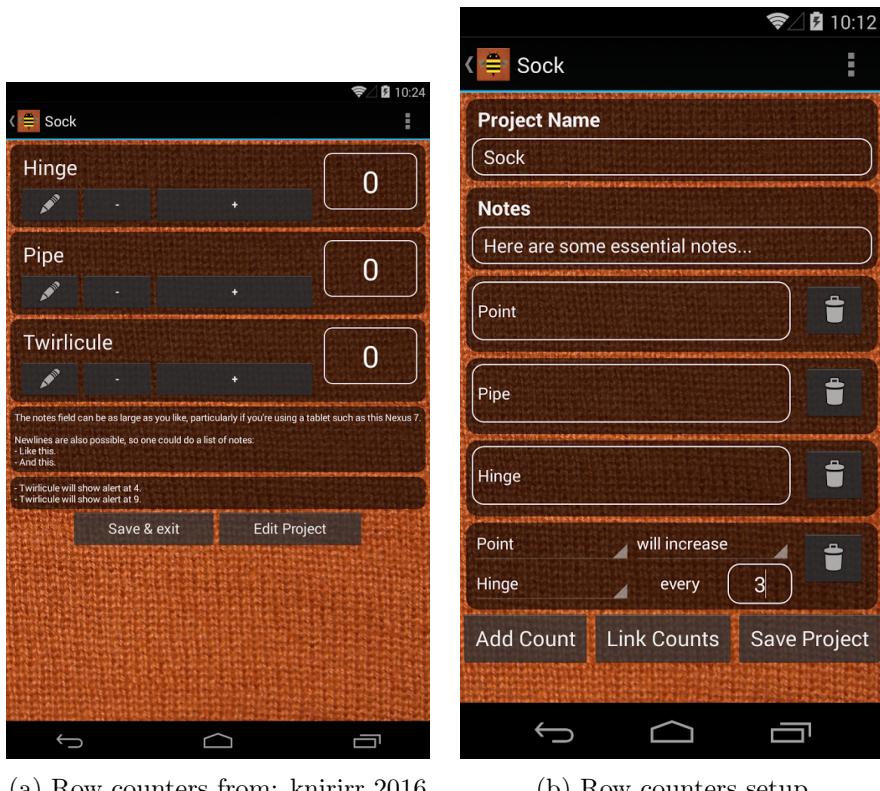


Figure 2.3: Screenshots of the app Knitting and Crochet Buddy

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 different counters to be triggered once the counter reaches a set number.



(a) Row counters from: knirrr 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.

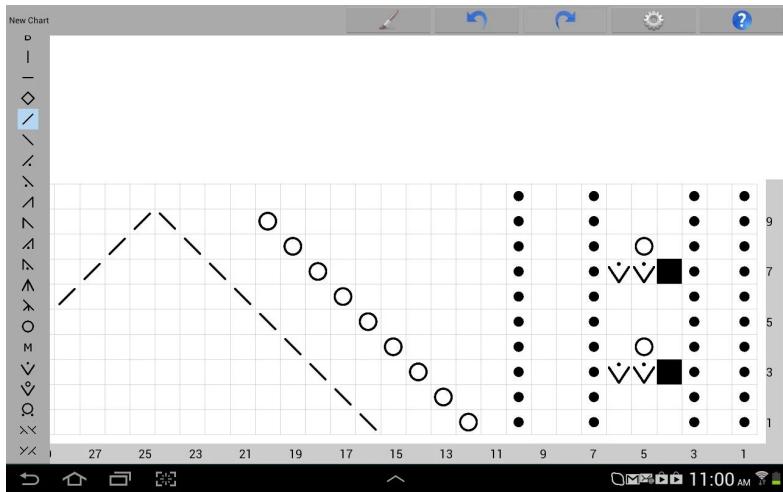


Figure 2.5: Chart editor of Knitting Chart Maker

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 mention 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 expressions¹. He aims to do so by using the Knitting Expression Langauge, KEL, that he defined (Whitall 2009). KEL is based on the Groovy programming language² for the Java platform and the GroovyMarkup architecture.

The following KEL expression

```
1 Pattern {
2     generalInformation
3 }
```

Listing 2.1: Example expression in KnitML

would result in

```
1 <pattern>
2     <general-information/>
3 </pattern>
```

Listing 2.2: Example expression in KnitM: XML result

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

2.3 UI Evaluation

As stated in the introduction (Chapter 1), ‘this thesis’ goal is to produce a prototype with a useful UI. 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 utility 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).

¹<http://www.knitml.com/blog/static.php?page=about-knitml>

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 A.4. From these interviews user stories were formulated and corresponding functional requirements were extracted — see table 3.1 below.

#	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	Create, Read, Update, Delete (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
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 prototype 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 must have good usability and be robust, meaning it should be able to handle errors without crashing. Pattern files must be able to be backed up locally and be accessible by the user. All prototype functionalities 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 like in 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

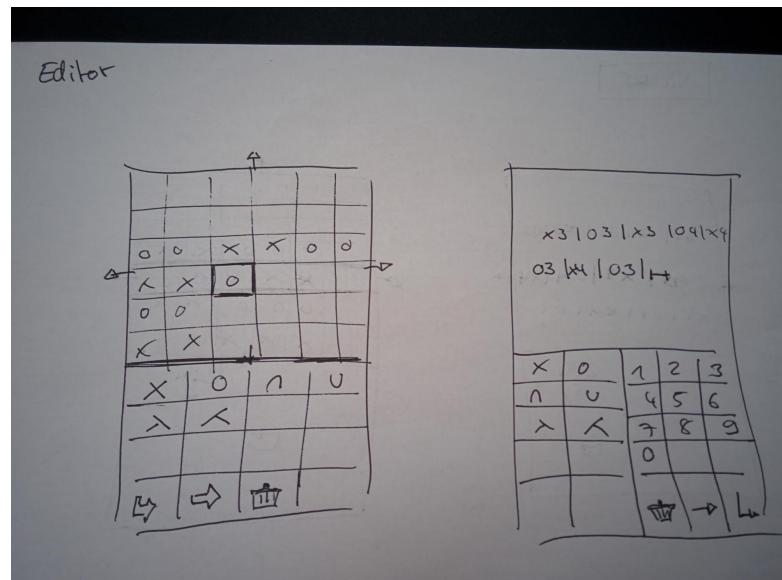


Figure 4.1: Editor screens for grid and row format

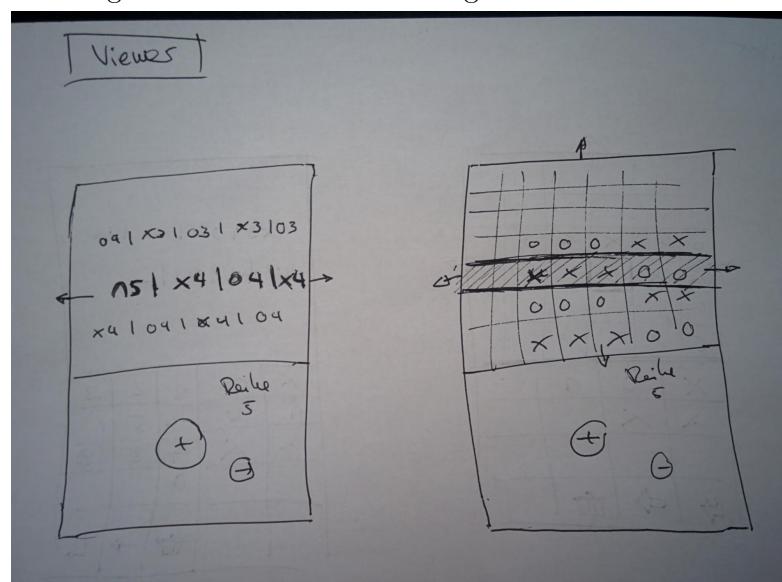


Figure 4.2: Viewer screens for grid and editor format with row counter

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.

Early concept sketches of the editor and viewer can be seen in **Figure 4.1** and **Figure 4.2**.

Chapter 5

Android Basics

5.1 The Operating System Android

Android is an open source operating system for mobile phones and tablets 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 reference¹, training², and Application Programming Interface (API) guides³ 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 fragments (see Section 5.2.3) — 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.

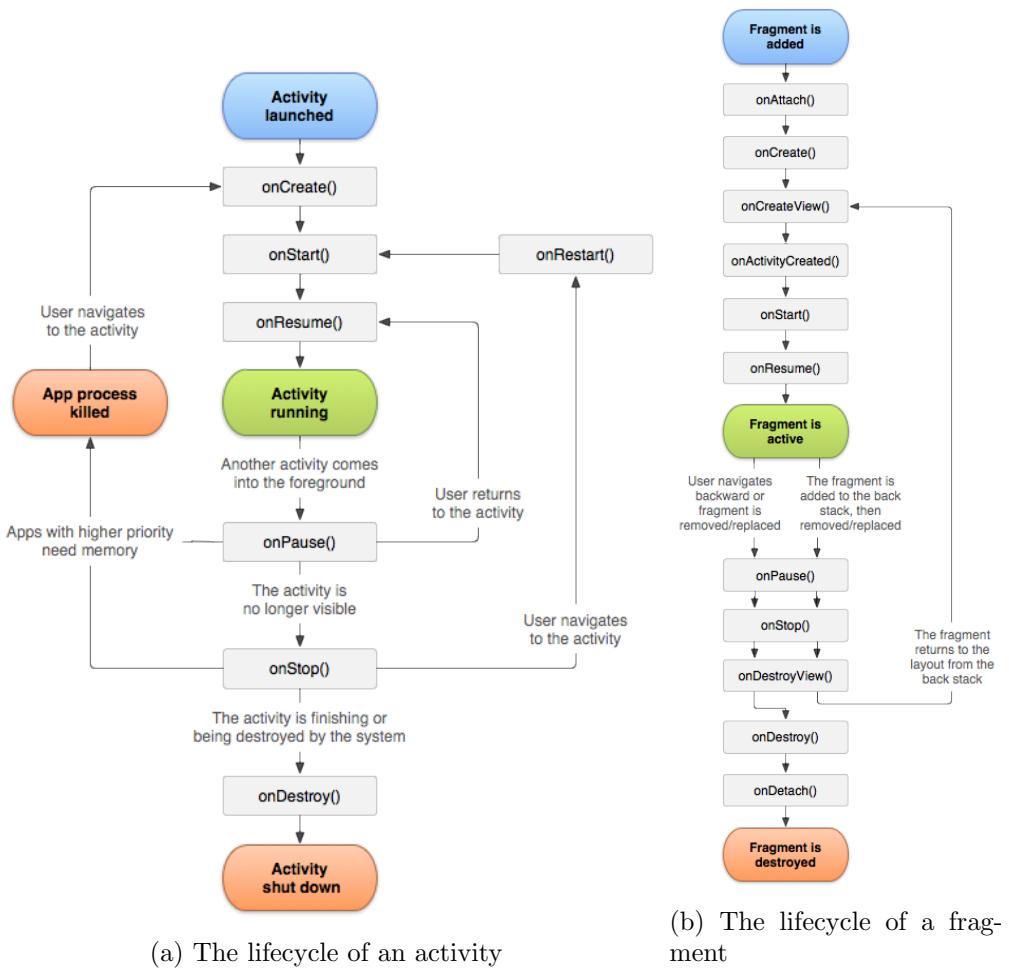


Figure 5.1: The lifecycles of activities and fragments

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 common 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 accidentally 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 intended 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**).

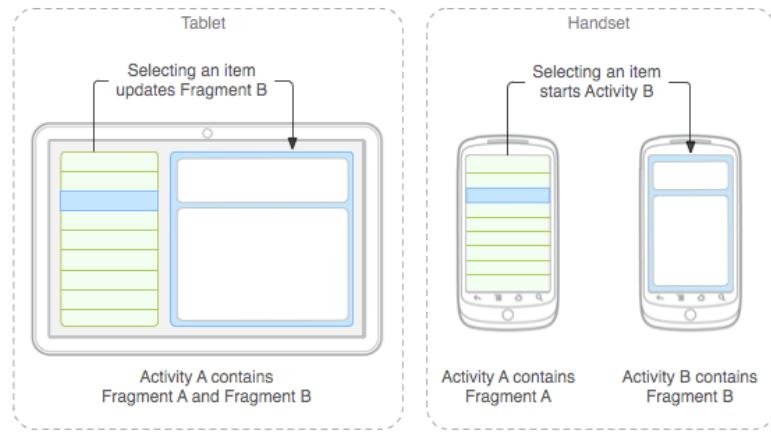


Figure 5.2: Two fragments of one activity and their layout on two different screen sizes

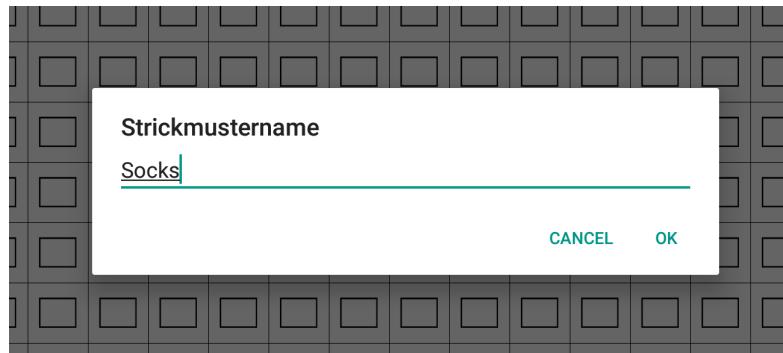


Figure 5.3: A dialog fragment for naming a pattern

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 practice 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
11                 implement OnArticleSelectedListener");
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 displays 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 hierarchy 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 Android devices often having 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 originally places them in the internal 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 originated 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, an 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 version need to request and acquire 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, except for 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 unusable 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 FontForge¹ was used creation of the custom TTF knitting font used in the prototype. 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.

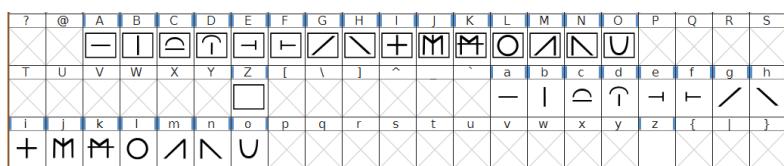


Figure 6.1: The custom knitting font used in the prototype

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** (see Section 6.4.1), and a single string with linefeeds used in the row format's **EditText** widget (see Section 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 prototype 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 permanent 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

²<https://github.com/google/gson> (last accessed: 2016-08-09)

maintaining inheritance hierarchies. The corresponding code can be found in the class `PatternStorage.java` (see Appendix B.26).

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 lists³, 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

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

should scale and scroll together with the grid, but should not be scrolled outside the 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 be 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 match 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 path⁶ present an implementation example and were used as a guideline for the implementation of the class `PatternGridView.java` (see Appendix B.20). 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, calculating the position of the corresponding 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 dragging 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

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)

speed and duration exponential to the velocity of the fling gesture, where the duration is calculated by introducing a friction to the scroll. This gesture can also be described as a swiping motion performed on the touchscreen. Even after the finger has been lifted off 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 initialized 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 subtracted 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:

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

$$\begin{aligned} offset_{max} = & (width_{view} - width_{content} - 2 * margin, \\ & height_{view} - height_{content} - 2 * margin) \end{aligned}$$

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

Similarly 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. The scale factor is then used during the drawing operation to scale the grid lines, axes, and text. When touching the grid the touched cell is calculated from the pixel position of the touch event. The currently selected stitch string is then saved to that position in the two-dimensional

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)

pattern array. Following this the view is invalidated¹⁰ and re-drawn with the updated pattern.

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` widget¹¹ fulfills most of these requirements. The `EditText` widget inherits from the class `EditText` which is specialised for displaying text content. It supports standard text editing functions, 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 activity's main window has input focus (Android Developers 2016c). An activity receives input focus on activity start and resume when containing an `EditText` widget in its view hierarchy. This behavior can be suppressed by declaring the state of the soft input method in the manifest file (see Appendix B.1) of the project as hidden (see *Listing 6.1*).

```

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

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

¹¹[`https://developer.android.com/reference/android/widget/EditText.html`](https://developer.android.com/reference/android/widget/EditText.html)

avoid this the custom widget `KeyboardlessEditText`¹², written by Danial Goodwin, is used in the prototype. It offers the same functions as a native Android `EditText` widget, but will suppress the showing of the on-screen keyboard.

To improve the visibility of the lines the `LinedEditor` (see Appendix B.15) class is subclassed from the keyboardless `EditText` widget and a background is drawn behind every other line. The `LinedEditor` is part of the `RowEditorLinearLayout` (see Appendix B.29), a custom `LinearLayout` which houses the complete row format editing functionality. This layout will be referred to as the row editor. A `LineNumberTextView` (see Appendix B.16), a custom `TextView`, is placed to the left of the `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 object 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 prototype 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

¹²<https://github.com/danialgoodwin/android-widget-keyboardless-edittext>

¹³<https://developer.android.com/reference/android/widget/Scroller.html>

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

Additionally, the timing of measurements are also the cause for 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 before 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 calculating the cursor's new position by increasing the value of its x-coordinates by the width of added text, the returned position is located 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 displayed at the correct location and not wrapped to the following line. This presents a problem when scrolling to offscreen text changes at the end 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.

Lastly, the line numbers do not stay visible when the row editor is scrolled horizontally which reduces its usability since the user cannot at all times tell the line number of a row. 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 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 the Android Open Source Project’s Calculator project project¹⁵. 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 (see Appendix B.30). The activity’s layout file defines a container `FrameLayout`¹⁶ for the

¹⁴<https://developer.android.com/reference/android/widget/GridView.html>

¹⁵https://android.googlesource.com/platform/packages/apps/Calculator/+/refs/tags/android-6.0.1_r7/src/com/android/calculator2/CalculatorPadLayout.java

¹⁶<https://developer.android.com/reference/android/widget/FrameLayout.html>

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 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` (see Appendix B.5) contains, just like the `ViewerActivity`, a `FrameLayout` 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, offering to save the changes or to discard them and close the editor. After a pattern is exported an info dialog displays 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` (see Appendix B.9), the `PatternDeleteDialogFragment` (see Appendix B.19), and the `PatternNameDialogFragment` (see Appendix B.23).

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` (see Appendix B.21) 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

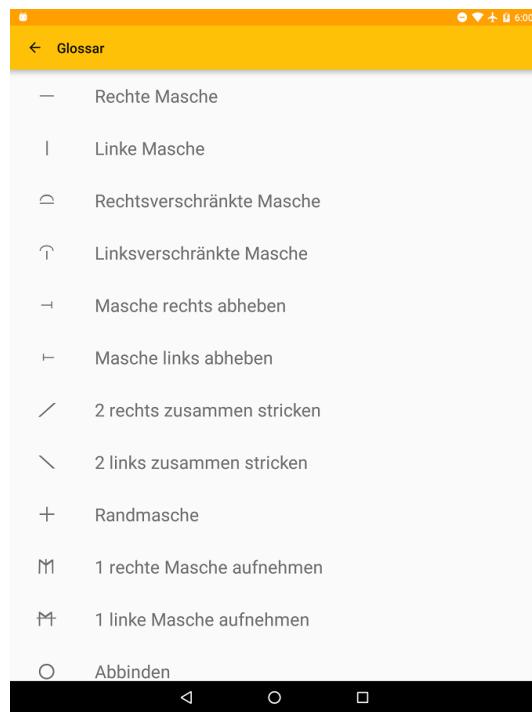
¹⁷<https://developer.android.com/reference/android/widget/ListView.html>

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.

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.

¹⁸see footnote 17



(a) The glossary

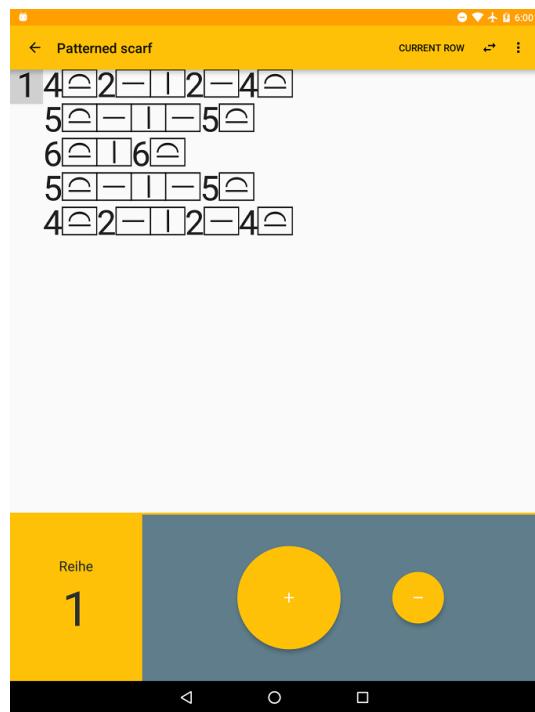
The screenshot shows a grid editor screen with a yellow header bar containing a back arrow and the text "Fingerless gloves". To the right of the header is a "GRID SIZE" button with a dropdown menu showing "6x6". The main area is a 10x10 grid of symbols, with rows and columns numbered 1 to 10. The symbols represent a pattern for fingerless gloves:

	1	2	3	4	5	6	7	8	9	10
1	\		—		—		—		—	/
2	—	\	—		—		—		/	
3	—		\		—		—	/	—	
4	—		—	\	—		/		—	
5	—		—		\	/	—		—	
6	—		—		/	\	—		—	
7	—		—	/	—		\		—	
8	—		/		—		—	\	—	
9	—	/	—		—		—		\	
10	/		—		—		—		—	

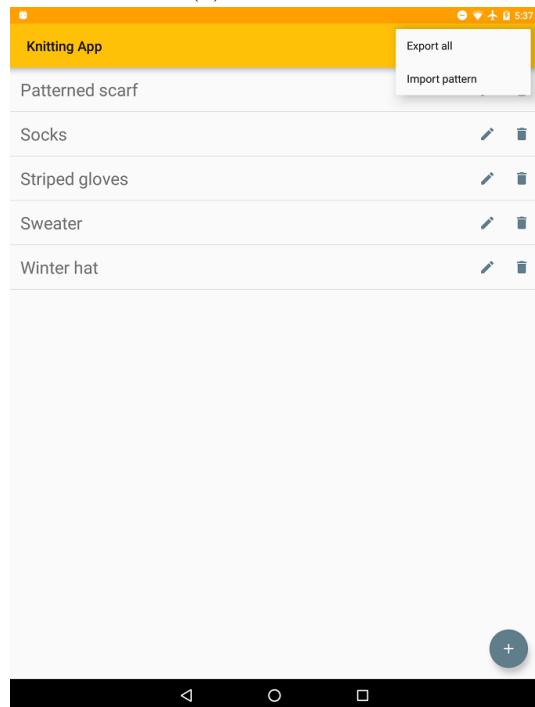


(b) The grid editor while showing a symbol description

Figure 6.2: Screenshots of the current version of the prototype



(a) The viewer



(b) The list of stored patterns

Figure 6.3: Screenshots of the current version of the prototype

Chapter 7

User Test

As stated in the introduction (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 prototype 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 whether the prototype met the expectations 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. A brief explanation on what the number and symbol combination signified in the row format and how it would be displayed in an expanded form when viewed in the grid format was necessary. After that explanation 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 fully grasp 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 functionalities 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.

After finishing the 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 keyboard 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 happened 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 prototype 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 repetitions 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.

After the feedback from the user tests the following changes were implemented and can now be found in the current prototype:

- The editor and viewer show the grid format first
- The grid editor sets the first symbol on the keyboard as active

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 to differentiate the line numbers, highlight current row)
5. Symbol descriptions and glossary are German only
6. Grid format has no button to reset the zoom

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 a challenge, 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 more user feedback would needed to be implemented. The current version of 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.

The requirements (see Chapter 3.2) that have not been fulfilled yet can be added to future versions of the app. Additionally, support of stitches that are wider than one cell can be added for knitting techniques that span across multiple stitches, e.g. the instruction to knit two together (k2tog). During the user tests one participant also wished for a repeat counter to be included in the viewer, for cases where a certain pattern has to be repeated, e.g. as is often done in scarves. Other features could be to control the app with voice commands or to have a pattern read aloud to the user.

It is also possible to integrate the functions of the prototype into an app designed to manage everything connected to knitting projects. Such an 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.

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

1.1	Knitting patterns and their corresponding pattern charts from Natter 1983, p142	2
2.1	Screenshots of the app knit tink at: https://play.google.com/store/apps/details?id=com.warrencollective.knittink (last accessed: 2016-08-08)	5
a	Row counter at: https://lh6.ggpht.com/-9DvA3pUKqPQwDwi8P_mZX0EhyKz9pE4Dks2QuEKxEGJePvXfY4hUkL00i-zud38c5Y=h900-rw (last accessed: 2016-04-04)	5
b	Project setup from: Warren 2015	5
2.2	Screenshots of the app Knitting Counter at: https://play.google.com/store/apps/details?id=org.kuklake.rowCounter (last accessed: 2016-08-08)	6
a	Row counter at: https://lh4.ggpht.com/M05RYCkE7md51ckjB9Bf_CQjz-L6fSS3aWFnQ8UAoURXj04BuHZiWeHtImzAhhpvekE=h900-rw (last accessed: 2016-04-04)	6
b	Row counter setup at: https://lh3.ggpht.com/AuzeRh7n_r4yLpk9puanH0pBDhcxj6AwC8h5qCaMN3TsRMRu7rML9awuPZTf49M_ejo=h900-rw (last accessed: 2016-04-04)	6
2.3	Screenshots of the app Knitting and Crochet Buddy at: https://play.google.com/store/apps/details?id=androiddeveloperjoe.knittingbuddy (last accessed: 2016-08-08)	7
a	Row counter with pattern chart picture at: https://lh3.ggpht.com/KJfgkhsUvqPCJSxqd7Tf09gVRgivtng8nfHgUENAHx401J-EqgPvTbCMW-dTrWVqzJE=h900-rw (last accessed: 2016-04-04)	7

b	Row counter with written pattern instructions at: https://lh3.ggpht.com/EPs72ilPpGCF_fMckHsVb2LeYVx-p6eNjcqg69e0wlsS2h0neneEMpH29CYH3rEM_c_=h900-rw (last accessed: 2016-04-04)	7
2.4	Screenshots of the app BeeCount Knitting Counter at: https://play.google.com/store/apps/details?id=com.knirirr.beeccount (last accessed: 2016-08-08)	8
a	Row counters from: knirirr 2016	8
b	Row counters setup from: https://lh4.ggpht.com/ZD3ujRmMgBuxEaDjnCsc9fcN9k_kUQYwfEr_mQ23n7t-0sg-arQOMMC-I52MI7ujc94=h900-rw (last accessed: 2016-04-04)	8
2.5	Chart editor of Knitting Chart Maker at: https://lh6.ggpht.com/MGKM0ukCD1MWWuboyxmZT-y8P3fTha4SI617u31eK3jFIkLsAllNEA_g6NffaoKRqyg=h900-rw (last accessed: 2016-04-04)	9
4.1	Editor screens for grid and row format (own image)	16
4.2	Viewer screens for grid and editor format with row counter (own image)	16
5.1	The lifecycles of activities and fragments	18
a	The lifecycle of an activity at https://developer.android.com/images/activity_lifecycle.png (last accessed: 2016-04-04)	18
b	The lifecycle of a fragment at: https://developer.android.com/images/fragment_lifecycle.png (last accessed: 2016-08-09)	18
5.2	Two fragments of one activity and their layout on two different screen sizes. at: https://developer.android.com/images/fundamentals/fragments.png (last accessed: 2016-08-09)	20
5.3	A dialog fragment for naming a pattern (own image)	20
6.1	The custom knitting font used in the prototype (own image)	24
6.2	Screenshots of the current version of the prototype	36
a	The glossary (own image)	36

b	The grid editor while showing a symbol description (own image)	36
6.3	Screenshots of the current version of the prototype	37
a	The viewer (own image)	37
b	The list of stored patterns (own image)	37

Listings

2.1	Example expression in KnitML	10
2.2	Example expression in KnitM: XML result	10
5.1	Example code for enforcing the implementation of a callback interface . .	20
6.1	Declaring on-screen keyboard hidden in manifest file.	29
6.2	Excerpt from row`editor.xml	30

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).
 - *jactivity*. 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).
- knirrr. *BeeCount Knitting Counter*. May 8, 2016. URL: https://lh5.ggpht.com/CaLXmsrgU6JmB1iswLwffjY2eMf0gtt90H41RgHRmGPqro6XCrMdnkawc_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/FfgM0tcw2WerHBjS1eqm8NsjuxnTcfPHvxenf-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 Zel- lenansicht. 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 An- weisung 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 einzelne 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

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/
3   res/android">
4     package="de.muffinworks.knittingapp">
5       <uses-permission android:name="android.permission.
6         WRITE_EXTERNAL_STORAGE"/>
7       <application
8         android:allowBackup="true"
9         android:icon="@mipmap/ic_launcher"
10        android:label="@string/app_name"
11        android:supportRtl="true"
12        android:theme="@style/AppTheme">
13
14       <activity android:name=".ViewerActivity"/>
15
16       <activity android:name=".EditorActivity"
17         android:windowSoftInputMode="
```

Listing B.1: AndroidManifest.xml

```
1 package de.muffinworks.knittingapp ;
2 import android.Manifest ;
3 import android.app.Dialog ;
4 import android.content.DialogInterface ;
5 import android.content.IntentInfo ;
6 import android.content.pm.PackageManager ;
7 import android.content.res.Configuration ;
8 import android.os.Bundle ;
9 import android.support.annotation.Nullable ;
10 import android.support.v7.app.ActionBar ;
11 import android.support.v7.app.AlertDialog ;
12 import android.support.v7.app.AppCompatActivity ;
13 import android.view.Menuitem ;
14 import de.muffinworks.knittingapp.storage.PatternStorage ;
15 import de.muffinworks.knittingapp.util.Constants ;
16 public abstract class BaseActivity extends
17     AppCompatActivity {
18     protected String TAG = this .getClassName ()
19     () ;
20     protected String getSimpleName ()
21     () ;
22     protected PatternStorage mStorage ;
23     private ActionBar mActionBar ;
24     private AlertDialog mDialog ;
25     @Override
26     protected void onCreate(@Nullable Bundle
27         savedInstanceState) {
28         super.onCreate(savedInstanceState) ;
29         mStorage = PatternStorage .getInstance () ;
30         mStorage .init (this ) ;
31         mActionBar = getSupportFragmentManager () ;
32         setRequestedOrientation(ActivityInfo .
33             SCREEN_ORIENTATION_SENSOR_PORTRAIT) ;
34     }
35     @Override
36     protected void enableBackInActionBar(boolean
37         enabled)
38     ) {
39         mActionBar .setDisplayHomeAsUpEnabled (enabled) ;
40         mActionBar .setDisplayShowHomeEnabled (enabled) ;
41     }
42     @Override
43     public boolean onOptionsItemSelected(MenuItem item)
44     {
45         if (item .getItemId () == android.R .id .home) {
46             onBackPressed () ;
47         }
48     }
49     return super .onOptionsItemSelected (item) ;
50     protected void setActionBarTitle (String title)
51     {
52         mActionBar .setTitle (title) ;
53     }
54     protected boolean isExternalStoragePermissionGranted
55     () {
56         return checkSelfPermission (Manifest .permission .
57             WRITE_EXTERNAL_STORAGE)
58         == PackageManager.PERMISSION_GRANTED ;
59     }
60     @Override
61     protected void onStop()
62         super .onStop () ;
63         if (mDialog != null) mDialog .dismiss () ;
64     }
65     // https://developer.android.com/training/permissions
66     // requesting.html
67     protected void requestExternalStoragePermission()
68     {
69         if (!isExternalStoragePermissionGranted ())
70             if (shouldShowRequestPermissionRationale (
71                 Manifest .permission .
```

```

    WRITE_EXTERNALSTORAGE) {
        /* 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_EXTERNALSTORAGE},
                    Constants.PERMISSION_REQUEST_WRITE_SD});
        });
        return;
    }
    requestPermissions(new String[]{Manifest.permission.WRITE_EXTERNALSTORAGE},
        permission.WRITE_EXTERNALSTORAGE,
        Constants.PERMISSION_REQUEST_WRITE_SD);
}

```

```

    */
    * Copyright (C) 2014 The Android Open Source Project
    * Licensed under the Apache License, Version 2.0 (the "License");
    * you may not use this file except in compliance with
    * the License.
    * You may obtain a copy of the License at
    * http://www.apache.org/licenses/LICENSE-2.0
    * Unless required by applicable law or agreed to in
    * writing, software distributed under the License is distributed on an "AS IS" BASIS,
    * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    * express or implied.
    * See the License for the specific language governing
    * permissions and limitations under the License.
    */
    * https://github.com/rahulparsani/material-calculator2
    */
    package com.android.calculator2;
    import android.content.Context;
    import android.content.res.TypedArray;
    import android.support.v4.view.ViewCompat;
    import android.util.AttributeSet;
    import android.view.View;
    import android.view.ViewGroup;
    /*

```

```

    /**
     * should show permission
     * info_storage_permission,
     * new DialogInterface.OnClickListener
     () {
         @Override
         public void onClick(DialogInterface dialog, int
             which) {
             requestPermissions(new
                 String[]{Manifest.permission.WRITE_EXTERNALSTORAGE}),
                 Constants.PERMISSION_REQUEST_WRITE_SD});
         });
         return;
     }
     requestPermissions(new String[]{Manifest.permission.WRITE_EXTERNALSTORAGE},
         permission.WRITE_EXTERNALSTORAGE,
         Constants.PERMISSION_REQUEST_WRITE_SD);
}

```

```

    */
    * A layout that places children in an evenly
    * distributed grid based on the specified
    * {@link android.R.attr#columnCount} and {@link android
    * .R.attr#rowCount} attributes
    */
    public class CalculatorPadLayout extends ViewGroup {
        private int mRowCount;
        private int mColumnCount;
        private AttributeSet attrs;
        public CalculatorPadLayout(Context context) {
            this(context, null);
        }
        public CalculatorPadLayout(Context context,
            AttributeSet attrs) {
            this(context, attrs, 0);
        }
        public CalculatorPadLayout(Context context,
            AttributeSet attrs, int defStyle) {
            super(context, attrs, defStyle);
        }
        TypedArray a = context.obtainStyledAttributes(attrs,
            android.R.styleable.ColumnHeader,
            defStyle, 0);
        mRowCount = a.getInt(0, 1);
        mColumnCount = a.getInt(1, 1);
        a.recycle();
    }

```

```

    PERMISSION_REQUEST_WRITE_SD);
}

```

Listing B.2: BaseActivity.java

APPENDIX B. SOURCE CODE

```

56     final int childHeight = childView.getHeight();
57     childTop = childView.getMeasuredWidth();
58     if (childWidth != childView.getMeasuredWidth()
59         () || childHeight != childView.getMeasuredHeight()) {
60         childView.measure(
61             MeasureSpec.makeMeasureSpec(
62                 childWidth, MeasureSpec.EXACTLY),
63             MeasureSpec.makeMeasureSpec(
64                 childHeight, MeasureSpec.EXACTLY));
65     }
66 }
67
68 @Override
69 protected void onLayout(boolean changed, int left,
70     int top, int right, int bottom) {
71     final int columnWidth = Math.round((float) (right - left) /
72         paddingLeft - paddingRight) / mColumnCount;
73     final int rowHeight = Math.round((float) (bottom - top) /
74         paddingTop - paddingBottom) / mRowCount;
75     for (int childIndex = 0; childIndex <
76         getChildCount(); ++childIndex) {
77         final View childView = getChildAt(childIndex);
78         if (childView.getVisibility() == View.GONE)
79             continue;
80         final MarginLayoutParams lp = (
81             childView.getLayoutParams());
82         final int childTop = paddingTop + lp.
83             topMargin + childBottom * rowHeight;
84         final int childBottom = childTop - lp.
85             topMargin - lp.bottomMargin + rowHeight;
86         final int childLeft = paddingLeft + lp.
87             leftMargin + lp.rightMargin + columnWidth;
88         final int childWidth = childRight -
89         childLeft;
90     }
91 }
92
93     childView.setLayoutDirection(this) ==
94     LAYOUT_DIRECTION_RTL;
95     final int columnWidth = Math.round((float) (right - left) /
96         paddingLeft - paddingRight) / mColumnCount;
97     final int rowHeight = Math.round((float) (bottom - top) /
98         paddingTop - paddingBottom) / mRowCount;
99     final int rowIndex = (rowIndex + (columnIndex + 1)) %
100     mRowCount;
101    columnIndex = (columnIndex + 1) %
102     mColumnCount;
103 }
104
105     @Override
106     protected LayoutParams generateLayoutParams(
107         AttributeSet attrs) {
108         return new MarginLayoutParams(getContext(),
109             attrs);
110     }
111     @Override
112     protected LayoutParams generateDefaultLayoutParams()
113         ()
114     {
115         return new MarginLayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
116     }
117     @Override
118     protected ViewGroup.LayoutParams generateLayoutParams(ViewGroup.LayoutParams p) {
119         return new MarginLayoutParams(p);
120     }
121     @Override
122     protected boolean checkLayoutParams(ViewGroup.LayoutParams p) {
123         return p instanceof MarginLayoutParams;
124     }
125 }

```

Listing B.3: CalculatorPadLayout.java

```

10   OwnKnittingFont.ttf";
11   public static String METADATAFILENAME = "metadata.json";
12   public static final String EXPORTDIR = "KnittingPatterns";
13   // gets path to external storage that is user accessible and won't be deleted after app install
14   public static final String EXPORTFOLDERPATH =
15     Environment.getExternalStorageDirectory()
16       .getAbsolutePath() +
17       + "/"+EXPORTDIR;
18
19   public static int FILEPICKERREQUESTCODE = 2342;
20
21   public static final String EXTRA_PATTERN_ID = "de.muffinworks.EXTRA_PATTERN_ID";
22   public static final String EXTRAPATTERNDELETED = "de.muffinworks.EXTRA_PATTERN_DELETED";
23
24   public static final int PERMISSION_REQUEST_WRITE_SD
25     = 1337;
26
27   public static final int REQUEST_CODE_EDITOR = 1;
28   public static final int DEFAULT_ROWS = 10;
29   public static final int DEFAULT_COLUMNS = 10;
30
31   public static final int MAX_ROWS_AND_COLUMNS_LIMIT =
32     35;
33
34   public static final String[] DEFAULT_PATTERN = {
35     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
36     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
37     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
38     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
39     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
40     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
41     "\u2022 Constants.EMPTY_SYMBOL, "\u2022 10" +
42     "\u2022 static final String[] SYMBOLDESCRIPTIONS =
43     { "\u2022 Rechte Masche", "Linke Masche", "\u2022
44     "\u2022 Linksvorsch\u00e4nkte Masche", "\u2022
45     "\u2022 Masche rechts abheben", "\u2022 Masche links
46     "\u2022 2 links zusammen stricken", "\u2022 zusammen stricken", "\u2022
47     "\u2022 1 rechte Masche aufnehmen", "\u2022 Randmasche", "\u2022
48     "\u2022 1 linke Masche aufnehmen", "\u2022 Abbinden", "\u2022
49     "\u2022 Rechts neigende Zunahme", "\u2022 Zunahme", "\u2022
50     "\u2022 neigende Umlauf", "\u2022 Links
51     "\u2022 Umchlag", "\u2022 Leerer Platzhalter", "\u2022
52   };
53 }
```

Listing B.4: Constants.java

```

1 package de.muffinworks.knittingapp;
2 import android.app.Activity;
3 import android.content.DialogInterface;
4 import android.os.Bundle;
5 import android.support.annotation.Nullable;
6 import android.support.v4.app.FragmentManager;
7 import android.support.v4.app.FragmentManager;
8 import android.support.v4.app.FragmentTransaction;
9 import android.support.v7.app.AlertDialog;
10 import android.view.Menu;
11 import android.view.MenuItem;
12 import android.view.View;
13 import android.widget.Button;
14 import java.io.IOException;
15 import java.util.List;
16 import java.io.IOException;
17 import de.muffinworks.knittingapp.fragments.GridEditorFragment;
18 import de.muffinworks.knittingapp.fragments.GridSizeDialogFragment;
19 import de.muffinworks.knittingapp.fragments.PatternDeletedDialogFragment;
20 import de.muffinworks.knittingapp.fragments.PatternNameDialogFragment;
21 import de.muffinworks.knittingapp.fragments.PatternNameEditorFragment;
22 import de.muffinworks.knittingapp.fragments.RowEditorFragment;
23 import de.muffinworks.knittingapp.storage.models.Pattern
24 import de.muffinworks.knittingapp.util.Constants;
25 public class EditorActivity extends BaseActivity
26   implements PatternNameDialogListener,
27             OnPatternNameInteractionListener,
28             OnPatternDeleteInteractionListener,
29             GridSizeDialogFragment.OnGridSizeInteractionListener {
30   private FragmentManager mFragmentManager;
31   private RowEditorFragment mRowEditorFragment;
32   private GridEditorFragment mGridEditorFragment;
33   private FragmentContainer mFragmentContainer;
34   private int mFragmentContainerId = R.id
35   private MenuItem mMenuItemSetGridSize;
36   private Pattern mPattern;
37   private String mPatternId = null;
38   private boolean mWasEdited = false;
39
40 }
```

```

93
94     @Override
95     protected void onCreate(@Nullable Bundle
96         savedInstanceState) {
97         super.onCreate(savedInstanceState);
98         setContentView(R.layout.activity_editor);
99         enableBackInActionBar(true);
100        mPatternId = getIntent().getStringExtra(
101            Constants.EXTRA_PATTERN_ID);
102        if (mPatternId != null) {
103            mPattern = mStorage.load(mPatternId);
104            setActionBarTitle(mPattern.getName());
105        }
106        @Override
107        public void onBackPressed() {
108            AlertDialog dialog = new AlertDialog.Builder(this)
109                .setTitle(getString(R.string.dialog_save_changes))
110                .setPositiveButton(R.string.dialog_yes,
111                    new DialogInterface.OnClickListener() {
112                        @Override
113                        public void onClick(DialogInterface dialog, int which) {
114                            savePattern();
115                            setResult(Activity.RESULT_OK);
116                            finish();
117                        }
118                    })
119                    .setNegativeButton(R.string.dialog_no,
120                        new DialogInterface.OnClickListener() {
121                            @Override
122                            public void onClick(DialogInterface dialog, int which) {
123                                dialog.cancel();
124                                setResult(Activity.RESULT_CANCELED);
125                                finish();
126                            }
127                        })
128                    .create();
129        }
130        showBeforeExitDialog.show();
131    }
132    @Override
133    public void switchEditors() {
134        savePattern();
135        if (id == R.id.export_pattern) {
136            fm.replace(mFragmentContainer,
137                mGridEditorFragment);
138        } else if (id == R.id.open_glossary) {
139            fm.replace(mFragmentContainer,
140                mGlossaryActivity);
141        } else if (id == R.id.edit_pattern_name) {
142            fm.replace(mFragmentContainer,
143                mEditActivity);
144        }
145    }
146    @Override
147    public boolean onCreateOptionsMenu(Menu menu) {
148        getMenuInflater().inflate(R.menu.menu_editor,
149            menu);
150        MenuItem gridSize = menu.findItem(R.id.
151            mMenuItemSetGridSize);
152        gridSize.setTitle("true");
153        gridSize.setCheckable(true);
154        return true;
155    }
156    @Override
157    public boolean onOptionsItemSelected(MenuItem item) {
158        int id = item.getItemId();
159        if (id == R.id.set_size) {
160            showSetsizeDialog();
161        } else if (id == R.id.delete_pattern) {
162            showDeletePatternDialog();
163        } else if (id == R.id.switch_editor) {
164            switchEditors();
165        } else if (id == R.id.edit_pattern_name) {
166            showEditNameDialog();
167        } else if (id == R.id.save_pattern) {
168            savePattern();
169        } else if (id == R.id.open_glossary) {
170            startActivity(new Intent(
171                GlossaryActivity.class));
172        } else if (id == R.id.export_pattern) {
173            exportPattern();
174        }
175    }
176    private void exportPattern() {
177        try {
178            mStorage.export(mPatternId);
179            showAlertDialog(getString(R.string.
180                success_export_pattern,
181                Constants.EXPORT_DIR));
182        } catch (IOException e) {
183            e.printStackTrace();
184        }
185    }
186    private void switchEditors() {
187        startActivity(new Intent(
188            GlossaryActivity.class));
189    }
190    private void exportPattern() {
191        exportPattern();
192    }
193    private void savePattern() {
194        if (!mWasEdited) {
195            setResult(Activity.RESULT_OK);
196        } else {
197            setResult(Activity.RESULT_CANCELED);
198        }
199    }
200    private void showBeforeExitDialog() {
201        FragmentTransaction fm = mFragmentManager.
202            beginTransaction();
203        if (mRowEditorFragment.isVisible()) {
204            fm.replace(mFragmentContainer,
205                mGridEditorFragment);
206        }
207    }

```

```

138     mMenuItemSetGridSize.setVisibile (true) ;
139 } else {
140     fm.replace(mFragmentManager,
141     mRowEditorFragment);
142     fm.commit();
143 }
144 }
145 }
146 private boolean wasPatternEdited() {
147     if (mRowEditorFragment.isVisibile())
148         return mRowEditorFragment.hasPatternChanged()
149     } else {
150         return mGridEditorFragment.hasPatternChanged()
151     }
152 }
153 private void savePattern() {
154     if (wasPatternEdited())
155         if (mRowEditorFragment.isVisible())
156             mRowEditorFragment.savePattern();
157     } else {
158         mGridEditorFragment.savePattern();
159     }
160     mWasEdited = true;
161     mPattern = mStorage.load(mPatternId);
162 }
163 }
164 }
165 @Override
166 public void onSetChartSize(int columns, int rows) {
167     mGridEditorFragment.setGridSize(columns, rows);
168     savePattern();
169 }
170 }
171 public void showSetSizeDialog() {
172     GridSizeDialogFragment dialog =
173     GridSizeDialogFragment.newInstance(
174         mPattern.getColumns(),
175         mPattern.getRows());
176     dialog.show(mFragmentManager, getString(
177         R.string.tag_dialog_fragment_grid_size));
178 }
179 private void showEditNameDialog() {
180     Pattern pattern = mStorage.load(mPatternId);
181     PatternNameDialogFragment dialog =
182     PatternNameDialogFragment.newInstance(
183         pattern.getName());
184     dialog.show(mFragmentManager, getString(R.string
185         .tag_dialog_fragment_edit_name));
186 }
187 private void showDeletePatternDialog() {
188     PatternDeleteDialogFragment dialog =
189     PatternDeleteDialogFragment.newInstance(
190         mPattern.getName());
191     dialog.show(mFragmentManager, getString(R.string
192         .tag_dialog_fragment_delete_pattern));
193 }
194 public void onNumPadClick(View view) {
195     String num = ((Button) view).getText().toString();
196     mRowEditorFragment.onNumPadClick(num);
197 }
198 public void onDeleteToggled(View view) {
199     mGridEditorFragment.onDeleteToggled();
200 }
201 }
202 }
203 @Override
204 public void onSetName(String name) {
205     mPattern.setName(name);
206     mStorage.save(mPattern);
207     setActionBarTitle(mPattern.getName());
208     mWasEdited = true;
209     refreshFragmentData();
210 }
211 }
212 @Override
213 public void onConfirmDelete() {
214     mStorage.delete(mPatternId);
215     Intent resultIntent = new Intent();
216     resultIntent.putExtra(Constants.EXTRA_PATTERN_DELETED, true);
217     setResult(Activity.RESULT_CANCELED, resultIntent);
218 }
219 }
220 }
221 }

```

Listing B.5: EditorActivity.java

```

7 import android.widget.ListView;
8 import de.muffinworks.knittingapp;
9 import de.muffinworks.knittingapp.views.adapters.
10 public class GlossaryActivity extends BaseActivity {
11
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         mActionBar = getSupportActionBar();
21         mActionBar.setDisplayHomeAsUpEnabled(true);
22         mActionBar.setDisplayShowHomeEnabled(true);
23         mActionBar.setTitle(R.string.activity_title_glossary);
24
25         mGlossaryListView = (ListView) findViewById(R.id
26             .glossary_listview);

```

Listing B.6: GlossaryActivity.java

```

27         mGlossaryListView.setAdapter(new GlossaryAdapter
28             (this));
29
30     @Override
31     public boolean onOptionsItemSelected(MenuItem item) {
32         if (item.getItemId() == android.R.id.home) {
33             onBackPressed();
34         }
35     }
36 }
37

```

```

43     @Override
44     public View getView(int position, View convertView,
45         ViewGroup parent) {
46         GlossaryItemViewHolder viewHolder;
47
48         if (convertView == null) {
49             convertView = LayoutInflater.from(context)
50                 .inflate(R.layout.glossary_item_view_holder,
51                     null);
52             viewHolder = new GlossaryItemViewHolder(
53                 convertView);
54             viewHolder.mSymbol.setTypeface(Typeface
55                 .createFromAsset(mContext.getAssets(),
56                 Constants.KNITTING.FONT_PATH));
57             convertView.setTag(viewHolder);
58             viewHolder.mSymbol.setText(Constants.SYMBOLS[
59                 position]);
60             viewHolder.mDescription.setText(Constants.SYMBOLDESCRIPTIONS[position]);
61         }
62         return convertView;
63     }
64
65     static class GlossaryItemViewHolder {
66         public TextView mSymbol;
67         public TextView mDescription;
68
69         public GlossaryItemViewHolder(View root) {
70             mSymbol = (TextView) root.findViewById(R.id.
71                 mDescription);
72             mDescription = (TextView) root.findViewById(R.id.
73                 mSymbol_description);
74         }
75     }
76 }
77
78
79     @Override
80     public Object getItem(int position) {
81         return Constants.SYMBOLS[position];
82     }
83
84     @Override
85     public long getItemId(int position) {
86         return 0;
87     }
88
89     @Override
90     public boolean isEnabled(int position) {
91         return false;
92     }

```

Listing B.7: 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 import java.util.Arrays;
13 import java.util.List;
14 import de.muffinworks.knittingapp.R;
15 import de.muffinworks.knittingapp.storage.PatternStorage;
16 import de.muffinworks.knittingapp.storage.models.Pattern;
17 import de.muffinworks.knittingapp.storage.views.adapters.
18 import de.muffinworks.knittingapp.views.PatternGridView;
19 import de.muffinworks.knittingapp.views.adapters.
20 import de.muffinworks.knittingapp.views.adapters.
21 public class GridEditorFragment extends Fragment
22     implements KeyboardToggleAdapter.
23     GridEditorKeyListener {
24     private static final String BUNDLE_ID = "id";
25     private PatternStorage mStorage;
26     private PatternGridview mPatternGridview;
27     private PatternStorage mPattern;
28     private PatternGridview mKeyBoard;
29     private LinearLayout mDeleteButtonContainer;
30     private KeyboardToggleAdapter mKeyboardAdapter;
31     private boolean mIsDeleteActive = false;
32     private boolean mIsDelete;
33     public static GridEditorFragment getInstance(String
34         patternId) {
35         GridEditorFragment fragment = new
36             GridEditorFragment();
37         if (patternId != null) {
38             Bundle bundle = new Bundle();
39             bundle.putString(BUNDLE_ID, patternId);
40             fragment.setArguments(bundle);
41         }
42         return fragment;
43     }
44     @Override
45     public void onCreate(@Nullable Bundle
46         savedInstanceState) {
47         super.onCreate(savedInstanceState);
48         mStorage = PatternStorage.getInstance();
49         if (getArguments() != null) {
50             mPattern = mStorage.load(getString(BUNDLE_ID));
51         }
52     }
53     @Override
54     public View onCreateView(LayoutInflater inflater,
55         @Override
56         View view) {
57         @Nullable ViewGroup container;
58         return inflater.inflate(R.layout.
59             fragment_editor_grid, container, false);
60     }
61     @Override
62     public void onViewCreated(View view, @Nullable
63         Bundle savedInstanceState) {
64         super.onViewCreated(view, savedInstanceState);
65         mPatternGridView = (PatternGridView) view.
66         findViewById(R.id.gridView);
67         if (mPattern != null) {
68             mPatternGridView.setAdapter(mPattern.
69                 getPatternRows());
69         }
70         mDeleteButtonContainer = (LinearLayout) view.
71         findViewById(R.id.button_container);
72         mKeyboard = (GridView) view.findViewById(R.id.
73             keyboard_gridview);
74         mKeyboardAdapter = new KeyboardToggleAdapter(
75             getActivity(), this);
76         mKeyboard.setAdapter(mKeyboardAdapter);
77         //set first key active
78         mKeyboardAdapter.setOnClick();
79         mKeyboardAdapter.getAdapter().getView(0, null, null);
80     }
81     public void notifyDataSetChanged() {
82         if (mPattern != null) {
83             mPattern.notifyDataChanged();
84             String[] newPatternRows = mPattern.
85             getPattern();
86             mStorage.setPatternRows(newPatternRows);
87             Snackbar.make(mStorage.getView(), "Pattern
88             changed", Snackbar.LENGTH_SHORT).show();
89     }
90 }
91     public void savePattern() {
92         if (mPattern != null) {
93             String[] newPatternRows = mPattern.
94             getPattern();
95             mStorage.setPatternRows(newPatternRows());
96             mStorage.load(getString(BUNDLE_ID));
97         }
98     }
99     @Override
100    public void onKeyToggled(String key) {
101        setDeleteActive(false);
102    }
103 }

```

Listing B.8: GridEditorFragment.java

```

69         string dialog_title_grid_size() {
70             return dialog.create();
71         }
72     }
73
74     @Override
75     public Dialog onCreateDialog(Bundle savedInstanceState) {
76         LinearLayout content = (LinearLayout)
77             inflater().getLayoutInflater().inflate(R.layout.dialog_set_grid_size,
78                                         null);
79
80         mColumnsEditText = (EditText) content.
81             findViewById(R.id.edittext_columns);
82         mColumnsEditText.setText(Integer.toString(
83             mColumns));
84         mColumnsEditText.addTextChangedListener(new
85             DimensionTextWatcher(mColumnsEditText,
86             mColumns));
87
88         mRowsEditText = (EditText) content.findViewById(
89             R.id.edittext_rows);
90         mRowsEditText.setText(Integer.toString(mRows));
91         mRowsEditText.addTextChangedListener(new
92             DimensionTextWatcher(mRowsEditText, mRows));
93
94         mRowsEditText.setSelection(mRowsEditText.length
95         ());
96
97         AlertDialog dialog = new AlertDialog.Builder(
98             getActivity())
99             .setPositiveButton(R.string.dialog_ok,
100                 new DialogInterface.OnClickListener() {
101                     @Override
102                     public void onClick(DialogInterface dialog,
103                         int id) {
104                         dialog.dismiss();
105                     }
106                 });
107
108         private void onChartSizeSetResult(int columns,
109             int rows) {
110             mListener.onSetChartSize(columns, rows);
111         }
112
113         @Override
114         public void onAttach(Context context) {
115             super.onAttach(context);
116             if (context instanceof OnGridSizeInteractionListener) {
117                 mListener = (OnGridSizeInteractionListener)
118                     context;
119             }
120         }
121         @Override
122         public void runTimeException(context, String
123             error) {
124             throw new RuntimeException(error);
125         }
126
127         @Override
128         public void onDetach() {
129             super.onDetach();
130         }
131
132         public interface OnGridSizeInteractionListener {
133             void onSetChartSize(int columns, int rows);
134         }
135
136         class DimensionTextWatcher implements TextWatcher {
137             private int oldValue;
138             private EditText editText;
139
140             public DimensionTextWatcher(EditText editText,
141                 int oldValue) {
142                 this.editText = editText;
143                 this.oldValue = oldValue;
144             }
145
146             @Override
147             public void beforeTextChanged(CharSequence s,
148                 int start, int count, int after) {
149
150             @Override
151             public void onTextChanged(CharSequence s, int
152                 start, int before, int count) {
153
154             @Override
155             public void afterTextChanged(Editable s) {
156                 if (s.length() == 0) {
157                     ((AlertDialog) getDialog()).getButton(
158                         AlertDialog.BUTTON_POSITIVE).setEnabled(true);
159
160                 }
161             }
162
163             .setTitle(getResources().getString(R
164

```

```

157     setEnabled(false);
158 } else if (Integer.parseInt(s.toString()) == 167
159     0) { MAX_ROWS_AND_COLUMNS_LIMIT)
160     //input is 0
161     editText.setError(getString(R.string
162         error_dimension_zero));
163     editText.setSelection(editText.length());
164     ((AlertDialog)editDialog).getButton(
165         AlertDialog.BUTTON_POSITIVE).
166         setEnabled(false);
167 } else if (Integer.parseInt(s.toString()) > 171
168     Constants.MAX_ROWS_AND_COLUMNS_LIMIT) { 172
169     input > max_dimens 173
170     editText.setError(
171         getString(R.string
172             error_over_max_size,
173             Constants.
174             max_dimens));
175 }

```

Listing B.9: GridSizeDialogFragment.java

```

1 package de.muffinworks.knittingapp.views.adapters;
2 import android.content.Context;
3 import android.support.design.widget.Snackbar;
4 import android.view.LayoutInflater;
5 import android.widget.BaseAdapter;
6 import android.widget.ListView;
7 import de.muffinworks.knittingapp.util.Constants;
8 public abstract class KeyboardAdapterBase extends
9     BaseAdapter {
10     protected String[] mDescriptions;
11     protected String[] mCharacters;
12     static LayoutInflater inflater = null;
13     protected Context mContext;
14     protected Snackbar mSnackbar = null;
15     public KeyboardAdapterBase(Context context) {
16         mDescriptions = Constants.SYMBOL_DESCRIPTIONS;
17         mCharacters = Constants.SYMBOLS;
18         mContext = context;
19     }
20
21
22
23
24     inflater = (LayoutInflater)context.
25         getSystemService(Context.LAYOUT_INFLATER_SERVICE);
26
27     }
28     @Override
29     public int getCount() {
30         return mCharacters.length;
31     }
32     @Override
33     public Object getItem(int position) {
34         return mCharacters[position];
35     }
36     @Override
37     public long getItemId(int position) {
38         return position;
39     }
40
41
42
43

```

Listing B.10: KeyboardAdapterBase.java

```

1 /* 9 to use, copy, modify, merge, publish, distribute,
2 * The MIT License (MIT) 10 copies of the Software, and to permit persons to whom
3 * Copyright (c) 2014 Danial Goodwin (danialgoodwin.com) 11 the Software is
4 * furnished to do so, subject to the following conditions:
5 * Permission is hereby granted, free of charge, to any 12 The above copyright notice and this permission notice
6 * person obtaining a copy 13 shall be included in all
7 * of this software and associated documentation files (the 14 copies or substantial portions of the Software.
8 * in the Software without restriction, including without 15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF
9 * limitation the rights 16 ANY KIND, EXPRESS OR
10 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
11
12
13
14
15
16
17

```

```

18   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN      IN
19   NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
20   DAMAGES OR OTHER LIABILITY WHETHER IN AN ACTION OF CONTRACT, TORT OR
21   OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
22   OTHER DEALINGS IN THE SOFTWARE.
23 */
24 package net.simpleadvanced.widgets;
25 import android.content.Context;
26 import android.graphics.Rect;
27 import android.text.InputType;
28 import android.util.AttributeSet;
29 import android.util.Log;
30 import android.view.MotionEvent;
31 import android.view.View;
32 import android.view.inputmethod.InputMethodManager;
33 import android.widget.EditText;
34 import android.widget.TextView;
35 import android.widget.Widget;
36 import java.lang.reflect.InvocationTargetException;
37 import java.lang.reflect.Method;
38 import /**
39 * This is the same as a native EditText, except that no
40 * soft keyboard will appear when user clicks on widget. All other
41 * normal operations still work.
42 * To use in XML, add a widget for <my.package.name>.
43 * To use in Java, use one of the three constructors in
44 * this class
45 * To use in Java, use one of the three constructors in
46 * this class
47 */
48 public class KeyboardlessEditText2 extends EditText {
49     private static final Method mShowSoftInputOnFocus =
50         getMethod(EditText.class, "setShowSoftInputOnFocus",
51             boolean.class);
52     private OnClickListener mOnClickListener = new
53         OnClickListener() {
54             @Override
55             public void onClick(View v) {
56                 setCursorVisible(true);
57             }
58         };
59     private OnLongClickListener mOnLongClickListener =
60         new OnLongClickListener() {
61             @Override
62             public boolean onLongClick(View v) {
63                 setCursorVisible(false);
64             }
65         };
66     private KeyboardlessEditText2(Context context) {
67         super(context);
68         super(KeyboardlessEditText2.super(context));
69     }
70     initialize();
71 }
72     public KeyboardlessEditText2(Context context,
73         AttributeSet attrs) {
74         super(context, attrs);
75         initialize();
76     }
77     public KeyboardlessEditText2(Context context,
78         AttributeSet attrs, int defStyle) {
79         super(context, attrs, defStyle);
80     }
81     private void initialize() {
82         synchronized(this) {
83             setInputType(getInputType() | InputType.
84             TYPE_TEXT_FLAG_NO_SUGGESTIONS);
85             setFocusableInTouchMode(true);
86         }
87         // Needed to show cursor when user interacts
88         // with EditText so that the edit operations
89         // still work. Without the cursor, the edit
90         // operations won't appear.
91         setOnTouchListener(mOnLongClickListener);
92         setOnLongClickListener(mOnLongClickListener);
93         // setShowSoftInputOnFocus(false); // This is a
94         // hidden method in TextView.
95         // reflexSetShowSoftInputOnFocus(false); //
96         // Ensure that cursor is at the end of the input
97         // box when initialized. Without this, the
98         // cursor may be at index 0 when there is text
99         // added via layout XML.
100        setSelection(getText().length());
101    }
102    @Override
103    protected void onFocusChanged(boolean focused,
104        direction, Rect previouslyFocusedRect) {
105        super.onFocusChanged(focused, direction,
106        previouslyFocusedRect);
107    }
108    @Override
109    public boolean onTouchEvent(MotionEvent event) {
110        final boolean ret = super.onTouchEvent(event);
111        // Must be done after super.onTouchEvent()
112        hideKeyboard();
113        return ret;
114    }
115    private void hideKeyboard() {
116        final InputMethodManager imm = ((InputMethodManager)
117            getSystemService(Context.
118            SYSTEM_SERVICE));

```

```

119     INPUTMETHODSERVICE);
120     if (imm != null && imm.isActive(this)) {
121         imm.hideSoftInputFromWindow(
122             getWindowToken(), 0);
123     }
124     private void reflexSetShowSoftInputOnFocus(boolean
125         show) {
126         if (mShowSoftInputOnFocus != null) {
127             invokeMethod(mShowSoftInputOnFocus, this,
128                     show);
129         } else {
130             // Use fallback method. Not tested.
131             hideKeyboard();
132         }
133         /* Returns method if available in class or
134          * superclass (recursively),
135          * otherwise returns null.
136         */
137         public static Method getMethod(Class<?> cls, String
138             className, Class<?>... parametersType) {
139             while (cls != Object.class) {
140                 try {
141                     return sCIs.getDeclaredMethod(methodName
142                         .getDeclaredMethod(methodName);
143                     } catch (NoSuchMethodException e) {
144                         sCIs = sCIs.getSuperclass();
145                         if (sCIs == null) {
146                             throw null;
147                         }
148                     }
149                     /**
150                      * Returns if available, otherwise returns
151                      * null.
152                     */
153                     public static Object invokeMethod(Method method,
154                         Object receiver, Object... args) {
155                         try {
156                             return method.invoke(receiver, args);
157                         } catch (IllegalAccessException e) {
158                             Log.e("Safe invoke fail", "Invalid access",
159                                 e);
160                         } catch (InvocationTargetException e) {
161                             Log.e("Safe invoke fail", "Invalid target",
162                                 e);
163                         }
164                     }
165                     return null;
166                 }
167             }
168         }
169         public void setDeleteActive(boolean active) {
170             if (active) {
171                 mActiveKeyPosition = -1;
172                 notifyDataSetChanged();
173             }
174         }
175         public void setActiveKeyPosition(int position, View
176             convertView, ViewGroup parent) {
177             KnittingFontButton key;
178             if (convertView == null) {
179                 key = (KnittingFontButton) inflater.inflate(
180                     R.layout.view-grid-key, null);
181             } else {
182                 key = (KnittingFontButton) convertView;
183             }
184             key.setActive(mActiveKeyPosition == position);
185             key.setText(mCharacters[position]);
186             key.setOnLongClickListener(new View
187                 .OnLongClickListener() {
188                     @Override
189                     public void onLongClick(View v) {
190                         GridEditorAdapter.GridEditorKeyListener
191                             mListener = v.getListener();
192                         mListener.onLongClick(v);
193                     }
194                 });
195         }
196         public KeyboardToggleAdapter extends
197             KeyboardAdapterBase {
198             private int mActiveKeyPosition = -1;
199             private GridEditorKeyListener mListener;
200             public KeyboardToggleAdapter(Context context,
201                 GridEditorKeyListener listener) {
202                 super(context);
203                 mListener = listener;
204             }
205             public void setOnLongClickListener(OnLongClickListener
206                 mListener) {
207                 mListener.onLongClick(mListener);
208             }
209         }
210         public void setOnLongClickListener(OnLongClickListener
211                 mListener) {
212             mListener.onLongClick(mListener);
213         }
214     }
215     public void setOnLongClickListener(OnLongClickListener
216                 mListener) {
217             mListener.onLongClick(mListener);
218         }
219     }
220     public void setOnLongClickListener(OnLongClickListener
221                 mListener) {
222             mListener.onLongClick(mListener);
223         }
224     }
225     public void setDeleteActive(boolean active) {
226         if (active) {
227             mActiveKeyPosition = -1;
228             notifyDataSetChanged();
229         }
230     }
231     @Override
232     public View getView(int position, View
233             convertView, ViewGroup parent) {
234         KnittingFontButton key;
235         if (convertView == null) {
236             key = (KnittingFontButton) inflater.inflate(
237                 R.layout.view-grid-key, null);
238             key.setActive(mActiveKeyPosition == position);
239             key.setText(mCharacters[position]);
240             key.setOnLongClickListener(new View
241                 .OnLongClickListener() {
242                     @Override
243                     public void onLongClick(View v) {
244                         GridEditorAdapter.GridEditorKeyListener
245                             mListener = v.getListener();
246                         mListener.onLongClick(v);
247                     }
248                 });
249             mListener.onLongClick(mListener);
250         }
251     }
252 }

```

Listing B.11: KeyboardlessEditText2.java

```

46    @Override
47    public boolean onLongClick(View v) {
48        mSnackbar = Snackbar.make(v, Snackbar.
49            mDescriptions[position], LENGTHLONG)
50            .setAction(R.string.dialog_ok,
51                new View.OnClickListener() {
52                    @Override
53                    public void onClick(View v)
54                        mSnackbar.dismiss();
55                });
56            mSnackbar.show();
57            return true;
58        });
59    }

```

Listing B.12: KeyboardToggleAdapter.java

```

1 package de.muffinworks.knittingapp.views.adapters;
2 import android.content.Context;
3 import android.support.design.widget.Snackbar;
4 import android.view.View;
5 import android.view.ViewGroup;
6 import android.view.Window;
7 import android.view.WindowManager;
8 import de.muffinworks.knittingapp.R;
9 import de.muffinworks.knittingapp.views.
KnittingFontButton;
10 public class KeyboardTypingAdapter extends
KeyboardAdapterBase {
11     public interface RowEditorKeyListener {
12         void onKeyClicked(String key);
13     }
14     private RowEditorKeyListener mListener;
15     public KeyboardTypingAdapter(Context context,
16         RowEditorKeyListener listener) {
17         super(context);
18         mListener = listener;
19     }
20     @Override
21     public View getView(final int position, View
convertView, ViewGroup parent) {
22         KnittingFontButton key;
23         if (convertView == null) {
24             key = (KnittingFontButton) inflater.inflate(
25                 R.layout.view_grid_key, null);
26         } else {
27             key = (KnittingFontButton) convertView;
28         }
29         key.setOnClickListener(new View.OnClickListener() {
30             @Override
31             public void onClick(View v) {
32                 key.setText(mCharacters[position]);
33                 key.setOnLongClickListener(new View.
OnLongClickListener() {
34                     @Override
35                     public void onLongClick(View v) {
36                         mListener.onKeyToggled(mCharacters[
37                             position]);
38                         notifyDataSetChanged();
39                     }
40                 });
41             }
42         });
43         return key;
44     }
45     @Override
46     public void onKeyToggled(int position) {
47         key.setText(mCharacters[position]);
48         key.setOnLongClickListener(new View.
OnLongClickListener() {
49             @Override
50             public void onLongClick(View v) {
51                 mListener.onClick(v);
52             }
53         });
54         key.setOnClickListener(new View.OnClickListener() {
55             @Override
56             public void onClick(View v) {
57                 mListener.onKeyClicked(mCharacters[
58                     position]);
59             }
60         });
61     }
62 }

```

Listing B.13: KeyboardTypingAdapter.java

```

1 package de.muffinworks.knittingapp.views;
2
3 import android.content.Context;
4 import android.graphics.Typeface;
5 import android.util.AttributeSet;
6
7 import de.muffinworks.knittingapp.R;
8 import de.muffinworks.knittingapp.util.Constants;
9 import de.muffinworks.knittingapp.widget.Button;
10
11 public class KnittingFontButton extends Button {
12     public KnittingFontButton(Context context) {
13         super(context);
14         init(context);
15     }
16
17     public KnittingFontButton(Context context,
18         AttributeSet attrs) {
19         super(context, attrs);
20         init(context);
21     }
22
23     public KnittingFontButton(Context context,
24         AttributeSet attrs, int defStyleAttr) {
25         super(context, attrs, defStyleAttr);
26     }
27
28     @Override
29     protected void init(Context context) {
30         Typeface typeface = getKnittingTypeFace(context);
31         if (typeface != null) {
32             setTypeface(typeface);
33         }
34     }
35
36     @Override
37     protected void setActive(boolean mIsActive) {
38         if (mIsActive) {
39             setTextColor(getResources().getColor(R.color
40                 .colorPrimary, null));
41         } else {
42             setTextColor(getResources().getColor(R.color
43                 .keyboard_button_text_color, null));
44         }
45     }
46
47     @Override
48     protected void onDraw(Canvas canvas) {
49         super.onDraw(canvas);
50
51         Paint paint = new Paint();
52         paint.setAntiAlias(true);
53         paint.setTextSize(textSize);
54         paint.setColor(textColor);
55
56         String text = getText();
57         if (text != null) {
58             canvas.drawText(text, 0, 0, paint);
59         }
60     }
61
62     @Override
63     protected void onFocusChanged(boolean gainFocus,
64         int status, KeyEvent keyEvent) {
65         super.onFocusChanged(gainFocus, status, keyEvent);
66
67         if (gainFocus) {
68             requestFocus();
69             setSelection(0);
70         }
71     }
72
73     @Override
74     protected void onLayout(boolean changed, int left,
75         int top, int right, int bottom) {
76         super.onLayout(changed, left, top, right, bottom);
77
78         if (changed) {
79             updateTextSize();
80         }
81     }
82
83     private void updateTextSize() {
84         if (textSize != null) {
85             textSize = (float) (textSize * Constants.KNITTING_FONTSIZE_MULTIPLIER);
86         }
87     }
88
89     @Override
90     protected void onSizeChanged(int w, int h, int oldw,
91         int oldh) {
92         super.onSizeChanged(w, h, oldw, oldh);
93
94         if (w != oldw || h != oldh) {
95             updateTextSize();
96         }
97     }
98
99     @Override
100    protected void onMeasure(int widthMeasureSpec,
101        int heightMeasureSpec) {
102        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
103
104        int width = MeasureSpec.getSize(widthMeasureSpec);
105        int height = MeasureSpec.getSize(heightMeasureSpec);
106
107        if (width != 0 & height != 0) {
108            setMeasuredDimension(width, height);
109        }
110    }
111
112    @Override
113    protected void onDraw(Canvas canvas) {
114        super.onDraw(canvas);
115
116        if (text != null) {
117            canvas.drawText(text, 0, 0, paint);
118        }
119    }
120
121    @Override
122    protected void onFocusChanged(boolean gainFocus,
123        int status, KeyEvent keyEvent) {
124        super.onFocusChanged(gainFocus, status, keyEvent);
125
126        if (gainFocus) {
127            requestFocus();
128            setSelection(0);
129        }
130    }
131
132    @Override
133    protected void onLayout(boolean changed, int left,
134        int top, int right, int bottom) {
135        super.onLayout(changed, left, top, right, bottom);
136
137        if (changed) {
138            updateTextSize();
139        }
140    }
141
142    @Override
143    protected void onSizeChanged(int w, int h, int oldw,
144        int oldh) {
145        super.onSizeChanged(w, h, oldw, oldh);
146
147        if (w != oldw || h != oldh) {
148            updateTextSize();
149        }
150    }
151
152    @Override
153    protected void onMeasure(int widthMeasureSpec,
154        int heightMeasureSpec) {
155        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
156
157        int width = MeasureSpec.getSize(widthMeasureSpec);
158        int height = MeasureSpec.getSize(heightMeasureSpec);
159
160        if (width != 0 & height != 0) {
161            setMeasuredDimension(width, height);
162        }
163    }
164
165    @Override
166    protected void onDraw(Canvas canvas) {
167        super.onDraw(canvas);
168
169        if (text != null) {
170            canvas.drawText(text, 0, 0, paint);
171        }
172    }
173
174    @Override
175    protected void onFocusChanged(boolean gainFocus,
176        int status, KeyEvent keyEvent) {
177        super.onFocusChanged(gainFocus, status, keyEvent);
178
179        if (gainFocus) {
180            requestFocus();
181            setSelection(0);
182        }
183    }
184
185    @Override
186    protected void onLayout(boolean changed, int left,
187        int top, int right, int bottom) {
188        super.onLayout(changed, left, top, right, bottom);
189
190        if (changed) {
191            updateTextSize();
192        }
193    }
194
195    @Override
196    protected void onSizeChanged(int w, int h, int oldw,
197        int oldh) {
198        super.onSizeChanged(w, h, oldw, oldh);
199
200        if (w != oldw || h != oldh) {
201            updateTextSize();
202        }
203    }
204
205    @Override
206    protected void onMeasure(int widthMeasureSpec,
207        int heightMeasureSpec) {
208        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
209
210        int width = MeasureSpec.getSize(widthMeasureSpec);
211        int height = MeasureSpec.getSize(heightMeasureSpec);
212
213        if (width != 0 & height != 0) {
214            setMeasuredDimension(width, height);
215        }
216    }
217
218    @Override
219    protected void onDraw(Canvas canvas) {
220        super.onDraw(canvas);
221
222        if (text != null) {
223            canvas.drawText(text, 0, 0, paint);
224        }
225    }
226
227    @Override
228    protected void onFocusChanged(boolean gainFocus,
229        int status, KeyEvent keyEvent) {
230        super.onFocusChanged(gainFocus, status, keyEvent);
231
232        if (gainFocus) {
233            requestFocus();
234            setSelection(0);
235        }
236    }
237
238    @Override
239    protected void onLayout(boolean changed, int left,
240        int top, int right, int bottom) {
241        super.onLayout(changed, left, top, right, bottom);
242
243        if (changed) {
244            updateTextSize();
245        }
246    }
247
248    @Override
249    protected void onSizeChanged(int w, int h, int oldw,
250        int oldh) {
251        super.onSizeChanged(w, h, oldw, oldh);
252
253        if (w != oldw || h != oldh) {
254            updateTextSize();
255        }
256    }
257
258    @Override
259    protected void onMeasure(int widthMeasureSpec,
260        int heightMeasureSpec) {
261        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
262
263        int width = MeasureSpec.getSize(widthMeasureSpec);
264        int height = MeasureSpec.getSize(heightMeasureSpec);
265
266        if (width != 0 & height != 0) {
267            setMeasuredDimension(width, height);
268        }
269    }
270
271    @Override
272    protected void onDraw(Canvas canvas) {
273        super.onDraw(canvas);
274
275        if (text != null) {
276            canvas.drawText(text, 0, 0, paint);
277        }
278    }
279
280    @Override
281    protected void onFocusChanged(boolean gainFocus,
282        int status, KeyEvent keyEvent) {
283        super.onFocusChanged(gainFocus, status, keyEvent);
284
285        if (gainFocus) {
286            requestFocus();
287            setSelection(0);
288        }
289    }
290
291    @Override
292    protected void onLayout(boolean changed, int left,
293        int top, int right, int bottom) {
294        super.onLayout(changed, left, top, right, bottom);
295
296        if (changed) {
297            updateTextSize();
298        }
299    }
299
300    @Override
301    protected void onSizeChanged(int w, int h, int oldw,
302        int oldh) {
303        super.onSizeChanged(w, h, oldw, oldh);
304
305        if (w != oldw || h != oldh) {
306            updateTextSize();
307        }
308    }
309
310    @Override
311    protected void onMeasure(int widthMeasureSpec,
312        int heightMeasureSpec) {
313        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
314
315        int width = MeasureSpec.getSize(widthMeasureSpec);
316        int height = MeasureSpec.getSize(heightMeasureSpec);
317
318        if (width != 0 & height != 0) {
319            setMeasuredDimension(width, height);
320        }
321    }
322
323    @Override
324    protected void onDraw(Canvas canvas) {
325        super.onDraw(canvas);
326
327        if (text != null) {
328            canvas.drawText(text, 0, 0, paint);
329        }
330    }
331
332    @Override
333    protected void onFocusChanged(boolean gainFocus,
334        int status, KeyEvent keyEvent) {
335        super.onFocusChanged(gainFocus, status, keyEvent);
336
337        if (gainFocus) {
338            requestFocus();
339            setSelection(0);
340        }
341    }
342
343    @Override
344    protected void onLayout(boolean changed, int left,
345        int top, int right, int bottom) {
346        super.onLayout(changed, left, top, right, bottom);
347
348        if (changed) {
349            updateTextSize();
350        }
351    }
351
352    @Override
353    protected void onSizeChanged(int w, int h, int oldw,
354        int oldh) {
355        super.onSizeChanged(w, h, oldw, oldh);
356
357        if (w != oldw || h != oldh) {
358            updateTextSize();
359        }
360    }
361
362    @Override
363    protected void onMeasure(int widthMeasureSpec,
364        int heightMeasureSpec) {
365        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
366
367        int width = MeasureSpec.getSize(widthMeasureSpec);
368        int height = MeasureSpec.getSize(heightMeasureSpec);
369
370        if (width != 0 & height != 0) {
371            setMeasuredDimension(width, height);
372        }
373    }
374
375    @Override
376    protected void onDraw(Canvas canvas) {
377        super.onDraw(canvas);
378
379        if (text != null) {
380            canvas.drawText(text, 0, 0, paint);
381        }
382    }
383
384    @Override
385    protected void onFocusChanged(boolean gainFocus,
386        int status, KeyEvent keyEvent) {
387        super.onFocusChanged(gainFocus, status, keyEvent);
388
389        if (gainFocus) {
390            requestFocus();
391            setSelection(0);
392        }
393    }
394
395    @Override
396    protected void onLayout(boolean changed, int left,
397        int top, int right, int bottom) {
398        super.onLayout(changed, left, top, right, bottom);
399
400        if (changed) {
401            updateTextSize();
402        }
403    }
403
404    @Override
405    protected void onSizeChanged(int w, int h, int oldw,
406        int oldh) {
407        super.onSizeChanged(w, h, oldw, oldh);
408
409        if (w != oldw || h != oldh) {
410            updateTextSize();
411        }
412    }
412
413    @Override
414    protected void onMeasure(int widthMeasureSpec,
415        int heightMeasureSpec) {
416        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
417
418        int width = MeasureSpec.getSize(widthMeasureSpec);
419        int height = MeasureSpec.getSize(heightMeasureSpec);
420
421        if (width != 0 & height != 0) {
422            setMeasuredDimension(width, height);
423        }
424    }
425
426    @Override
427    protected void onDraw(Canvas canvas) {
428        super.onDraw(canvas);
429
430        if (text != null) {
431            canvas.drawText(text, 0, 0, paint);
432        }
433    }
434
435    @Override
436    protected void onFocusChanged(boolean gainFocus,
437        int status, KeyEvent keyEvent) {
438        super.onFocusChanged(gainFocus, status, keyEvent);
439
440        if (gainFocus) {
441            requestFocus();
442            setSelection(0);
443        }
444    }
445
446    @Override
447    protected void onLayout(boolean changed, int left,
448        int top, int right, int bottom) {
449        super.onLayout(changed, left, top, right, bottom);
450
451        if (changed) {
452            updateTextSize();
453        }
454    }
454
455    @Override
456    protected void onSizeChanged(int w, int h, int oldw,
457        int oldh) {
458        super.onSizeChanged(w, h, oldw, oldh);
459
460        if (w != oldw || h != oldh) {
461            updateTextSize();
462        }
463    }
463
464    @Override
465    protected void onMeasure(int widthMeasureSpec,
466        int heightMeasureSpec) {
467        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
468
469        int width = MeasureSpec.getSize(widthMeasureSpec);
470        int height = MeasureSpec.getSize(heightMeasureSpec);
471
472        if (width != 0 & height != 0) {
473            setMeasuredDimension(width, height);
474        }
475    }
476
477    @Override
478    protected void onDraw(Canvas canvas) {
479        super.onDraw(canvas);
480
481        if (text != null) {
482            canvas.drawText(text, 0, 0, paint);
483        }
484    }
485
486    @Override
487    protected void onFocusChanged(boolean gainFocus,
488        int status, KeyEvent keyEvent) {
489        super.onFocusChanged(gainFocus, status, keyEvent);
490
491        if (gainFocus) {
492            requestFocus();
493            setSelection(0);
494        }
495    }
496
497    @Override
498    protected void onLayout(boolean changed, int left,
499        int top, int right, int bottom) {
500        super.onLayout(changed, left, top, right, bottom);
501
502        if (changed) {
503            updateTextSize();
504        }
505    }
505
506    @Override
507    protected void onSizeChanged(int w, int h, int oldw,
508        int oldh) {
509        super.onSizeChanged(w, h, oldw, oldh);
510
511        if (w != oldw || h != oldh) {
512            updateTextSize();
513        }
514    }
514
515    @Override
516    protected void onMeasure(int widthMeasureSpec,
517        int heightMeasureSpec) {
518        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
519
520        int width = MeasureSpec.getSize(widthMeasureSpec);
521        int height = MeasureSpec.getSize(heightMeasureSpec);
522
523        if (width != 0 & height != 0) {
524            setMeasuredDimension(width, height);
525        }
526    }
527
528    @Override
529    protected void onDraw(Canvas canvas) {
530        super.onDraw(canvas);
531
532        if (text != null) {
533            canvas.drawText(text, 0, 0, paint);
534        }
535    }
536
537    @Override
538    protected void onFocusChanged(boolean gainFocus,
539        int status, KeyEvent keyEvent) {
540        super.onFocusChanged(gainFocus, status, keyEvent);
541
542        if (gainFocus) {
543            requestFocus();
544            setSelection(0);
545        }
546    }
547
548    @Override
549    protected void onLayout(boolean changed, int left,
550        int top, int right, int bottom) {
551        super.onLayout(changed, left, top, right, bottom);
552
553        if (changed) {
554            updateTextSize();
555        }
556    }
556
557    @Override
558    protected void onSizeChanged(int w, int h, int oldw,
559        int oldh) {
560        super.onSizeChanged(w, h, oldw, oldh);
561
562        if (w != oldw || h != oldh) {
563            updateTextSize();
564        }
565    }
565
566    @Override
567    protected void onMeasure(int widthMeasureSpec,
568        int heightMeasureSpec) {
569        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
570
571        int width = MeasureSpec.getSize(widthMeasureSpec);
572        int height = MeasureSpec.getSize(heightMeasureSpec);
573
574        if (width != 0 & height != 0) {
575            setMeasuredDimension(width, height);
576        }
577    }
578
579    @Override
580    protected void onDraw(Canvas canvas) {
581        super.onDraw(canvas);
582
583        if (text != null) {
584            canvas.drawText(text, 0, 0, paint);
585        }
586    }
587
588    @Override
589    protected void onFocusChanged(boolean gainFocus,
590        int status, KeyEvent keyEvent) {
591        super.onFocusChanged(gainFocus, status, keyEvent);
592
593        if (gainFocus) {
594            requestFocus();
595            setSelection(0);
596        }
597    }
598
599    @Override
600    protected void onLayout(boolean changed, int left,
601        int top, int right, int bottom) {
602        super.onLayout(changed, left, top, right, bottom);
603
604        if (changed) {
605            updateTextSize();
606        }
607    }
607
608    @Override
609    protected void onSizeChanged(int w, int h, int oldw,
610        int oldh) {
611        super.onSizeChanged(w, h, oldw, oldh);
612
613        if (w != oldw || h != oldh) {
614            updateTextSize();
615        }
616    }
616
617    @Override
618    protected void onMeasure(int widthMeasureSpec,
619        int heightMeasureSpec) {
620        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
621
622        int width = MeasureSpec.getSize(widthMeasureSpec);
623        int height = MeasureSpec.getSize(heightMeasureSpec);
624
625        if (width != 0 & height != 0) {
626            setMeasuredDimension(width, height);
627        }
628    }
629
630    @Override
631    protected void onDraw(Canvas canvas) {
632        super.onDraw(canvas);
633
634        if (text != null) {
635            canvas.drawText(text, 0, 0, paint);
636        }
637    }
638
639    @Override
640    protected void onFocusChanged(boolean gainFocus,
641        int status, KeyEvent keyEvent) {
642        super.onFocusChanged(gainFocus, status, keyEvent);
643
644        if (gainFocus) {
645            requestFocus();
646            setSelection(0);
647        }
648    }
649
650    @Override
651    protected void onLayout(boolean changed, int left,
652        int top, int right, int bottom) {
653        super.onLayout(changed, left, top, right, bottom);
654
655        if (changed) {
656            updateTextSize();
657        }
658    }
658
659    @Override
660    protected void onSizeChanged(int w, int h, int oldw,
661        int oldh) {
662        super.onSizeChanged(w, h, oldw, oldh);
663
664        if (w != oldw || h != oldh) {
665            updateTextSize();
666        }
667    }
667
668    @Override
669    protected void onMeasure(int widthMeasureSpec,
670        int heightMeasureSpec) {
671        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
672
673        int width = MeasureSpec.getSize(widthMeasureSpec);
674        int height = MeasureSpec.getSize(heightMeasureSpec);
675
676        if (width != 0 & height != 0) {
677            setMeasuredDimension(width, height);
678        }
679    }
680
681    @Override
682    protected void onDraw(Canvas canvas) {
683        super.onDraw(canvas);
684
685        if (text != null) {
686            canvas.drawText(text, 0, 0, paint);
687        }
688    }
689
690    @Override
691    protected void onFocusChanged(boolean gainFocus,
692        int status, KeyEvent keyEvent) {
693        super.onFocusChanged(gainFocus, status, keyEvent);
694
695        if (gainFocus) {
696            requestFocus();
697            setSelection(0);
698        }
699    }
699
700    @Override
701    protected void onLayout(boolean changed, int left,
702        int top, int right, int bottom) {
703        super.onLayout(changed, left, top, right, bottom);
704
705        if (changed) {
706            updateTextSize();
707        }
708    }
708
709    @Override
710    protected void onSizeChanged(int w, int h, int oldw,
711        int oldh) {
712        super.onSizeChanged(w, h, oldw, oldh);
713
714        if (w != oldw || h != oldh) {
715            updateTextSize();
716        }
717    }
717
718    @Override
719    protected void onMeasure(int widthMeasureSpec,
720        int heightMeasureSpec) {
721        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
722
723        int width = MeasureSpec.getSize(widthMeasureSpec);
724        int height = MeasureSpec.getSize(heightMeasureSpec);
725
726        if (width != 0 & height != 0) {
727            setMeasuredDimension(width, height);
728        }
729    }
730
731    @Override
732    protected void onDraw(Canvas canvas) {
733        super.onDraw(canvas);
734
735        if (text != null) {
736            canvas.drawText(text, 0, 0, paint);
737        }
738    }
739
740    @Override
741    protected void onFocusChanged(boolean gainFocus,
742        int status, KeyEvent keyEvent) {
743        super.onFocusChanged(gainFocus, status, keyEvent);
744
745        if (gainFocus) {
746            requestFocus();
747            setSelection(0);
748        }
749    }
749
750    @Override
751    protected void onLayout(boolean changed, int left,
752        int top, int right, int bottom) {
753        super.onLayout(changed, left, top, right, bottom);
754
755        if (changed) {
756            updateTextSize();
757        }
758    }
758
759    @Override
760    protected void onSizeChanged(int w, int h, int oldw,
761        int oldh) {
762        super.onSizeChanged(w, h, oldw, oldh);
763
764        if (w != oldw || h != oldh) {
765            updateTextSize();
766        }
767    }
767
768    @Override
769    protected void onMeasure(int widthMeasureSpec,
770        int heightMeasureSpec) {
771        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
772
773        int width = MeasureSpec.getSize(widthMeasureSpec);
774        int height = MeasureSpec.getSize(heightMeasureSpec);
775
776        if (width != 0 & height != 0) {
777            setMeasuredDimension(width, height);
778        }
779    }
780
781    @Override
782    protected void onDraw(Canvas canvas) {
783        super.onDraw(canvas);
784
785        if (text != null) {
786            canvas.drawText(text, 0, 0, paint);
787        }
788    }
789
790    @Override
791    protected void onFocusChanged(boolean gainFocus,
792        int status, KeyEvent keyEvent) {
793        super.onFocusChanged(gainFocus, status, keyEvent);
794
795        if (gainFocus) {
796            requestFocus();
797            setSelection(0);
798        }
799    }
799
800    @Override
801    protected void onLayout(boolean changed, int left,
802        int top, int right, int bottom) {
803        super.onLayout(changed, left, top, right, bottom);
804
805        if (changed) {
806            updateTextSize();
807        }
808    }
808
809    @Override
810    protected void onSizeChanged(int w, int h, int oldw,
811        int oldh) {
812        super.onSizeChanged(w, h, oldw, oldh);
813
814        if (w != oldw || h != oldh) {
815            updateTextSize();
816        }
817    }
817
818    @Override
819    protected void onMeasure(int widthMeasureSpec,
820        int heightMeasureSpec) {
821        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
822
823        int width = MeasureSpec.getSize(widthMeasureSpec);
824        int height = MeasureSpec.getSize(heightMeasureSpec);
825
826        if (width != 0 & height != 0) {
827            setMeasuredDimension(width, height);
828        }
829    }
830
831    @Override
832    protected void onDraw(Canvas canvas) {
833        super.onDraw(canvas);
834
835        if (text != null) {
836            canvas.drawText(text, 0, 0, paint);
837        }
838    }
839
840    @Override
841    protected void onFocusChanged(boolean gainFocus,
842        int status, KeyEvent keyEvent) {
843        super.onFocusChanged(gainFocus, status, keyEvent);
844
845        if (gainFocus) {
846            requestFocus();
847            setSelection(0);
848        }
849    }
849
850    @Override
851    protected void onLayout(boolean changed, int left,
852        int top, int right, int bottom) {
853        super.onLayout(changed, left, top, right, bottom);
854
855        if (changed) {
856            updateTextSize();
857        }
858    }
858
859    @Override
860    protected void onSizeChanged(int w, int h, int oldw,
861        int oldh) {
862        super.onSizeChanged(w, h, oldw, oldh);
863
864        if (w != oldw || h != oldh) {
865            updateTextSize();
866        }
867    }
867
868    @Override
869    protected void onMeasure(int widthMeasureSpec,
870        int heightMeasureSpec) {
871        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
872
873        int width = MeasureSpec.getSize(widthMeasureSpec);
874        int height = MeasureSpec.getSize(heightMeasureSpec);
875
876        if (width != 0 & height != 0) {
877            setMeasuredDimension(width, height);
878        }
879    }
880
881    @Override
882    protected void onDraw(Canvas canvas) {
883        super.onDraw(canvas);
884
885        if (text != null) {
886            canvas.drawText(text, 0, 0, paint);
887        }
888    }
889
890    @Override
891    protected void onFocusChanged(boolean gainFocus,
892        int status, KeyEvent keyEvent) {
893        super.onFocusChanged(gainFocus, status, keyEvent);
894
895        if (gainFocus) {
896            requestFocus();
897            setSelection(0);
898        }
899    }
899
900    @Override
901    protected void onLayout(boolean changed, int left,
902        int top, int right, int bottom) {
903        super.onLayout(changed, left, top, right, bottom);
904
905        if (changed) {
906            updateTextSize();
907        }
908    }
908
909    @Override
910    protected void onSizeChanged(int w, int h, int oldw,
911        int oldh) {
912        super.onSizeChanged(w, h, oldw, oldh);
913
914        if (w != oldw || h != oldh) {
915            updateTextSize();
916        }
917    }
917
918    @Override
919    protected void onMeasure(int widthMeasureSpec,
920        int heightMeasureSpec) {
921        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
922
923        int width = MeasureSpec.getSize(widthMeasureSpec);
924        int height = MeasureSpec.getSize(heightMeasureSpec);
925
926        if (width != 0 & height != 0) {
927            setMeasuredDimension(width, height);
928        }
929    }
930
931    @Override
932    protected void onDraw(Canvas canvas) {
933        super.onDraw(canvas);
934
935        if (text != null) {
936            canvas.drawText(text, 0, 0, paint);
937        }
938    }
939
940    @Override
941    protected void onFocusChanged(boolean gainFocus,
942        int status, KeyEvent keyEvent) {
943        super.onFocusChanged(gainFocus, status, keyEvent);
944
945        if (gainFocus) {
946            requestFocus();
947            setSelection(0);
948        }
949    }
949
950    @Override
951    protected void onLayout(boolean changed, int left,
952        int top, int right, int bottom) {
953        super.onLayout(changed, left, top, right, bottom);
954
955        if (changed) {
956            updateTextSize();
957        }
958    }
958
959    @Override
960    protected void onSizeChanged(int w, int h, int oldw,
961        int oldh) {
962        super.onSizeChanged(w, h, oldw, oldh);
963
964        if (w != oldw || h != oldh) {
965            updateTextSize();
966        }
967    }
967
968    @Override
969    protected void onMeasure(int widthMeasureSpec,
970        int heightMeasureSpec) {
971        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
972
973        int width = MeasureSpec.getSize(widthMeasureSpec);
974        int height = MeasureSpec.getSize(heightMeasureSpec);
975
976        if (width != 0 & height != 0) {
977            setMeasuredDimension(width, height);
978        }
979    }
980
981    @Override
982    protected void onDraw(Canvas canvas) {
983        super.onDraw(canvas);
984
985        if (text != null) {
986            canvas.drawText(text, 0, 0, paint);
987        }
988    }
989
990    @Override
991    protected void onFocusChanged(boolean gainFocus,
992        int status, KeyEvent keyEvent) {
993        super.onFocusChanged(gainFocus, status, keyEvent);
994
995        if (gainFocus) {
996            requestFocus();
997            setSelection(0);
998        }
999    }
999
1000   @Override
1001  protected void onLayout(boolean changed, int left,
1002      int top, int right, int bottom) {
1003      super.onLayout(changed, left, top, right, bottom);
1004
1005      if (changed) {
1006          updateTextSize();
1007      }
1008  }
1008
1009  @Override
1010  protected void onSizeChanged(int w, int h, int oldw,
1011      int oldh) {
1012      super.onSizeChanged(w, h, oldw, oldh);
1013
1014      if (w != oldw || h != oldh) {
1015          updateTextSize();
1016      }
1017  }
1017
1018  @Override
1019  protected void onMeasure(int widthMeasureSpec,
1020      int heightMeasureSpec) {
1021      super.onMeasure(widthMeasureSpec, heightMeasureSpec);
1022
1023      int width = MeasureSpec.getSize(widthMeasureSpec);
1024      int height = MeasureSpec.getSize(heightMeasureSpec);
1025
1026      if (width != 0 & height != 0) {
1027          setMeasuredDimension(width, height);
1028      }
1029  }
1030
1031  @Override
1032  protected void onDraw(Canvas canvas) {
1033      super.onDraw(canvas);
1034
1035      if (text != null) {
1036          canvas.drawText(text, 0, 0, paint);
1037      }
1038  }
1039
1040  @Override
1041  protected void onFocusChanged(boolean gainFocus,
1042      int status, KeyEvent keyEvent) {
1043      super.onFocusChanged(gainFocus, status, keyEvent);
1044
1045      if (gainFocus) {
1046          requestFocus();
1047          setSelection(0);
1048      }
1049  }
1049
1050  @Override
1051  protected void onLayout(boolean changed, int left,
1052      int top, int right, int bottom) {
1053      super.onLayout(changed, left, top, right, bottom);
1054
1055      if (changed) {
1056          updateTextSize();
1057      }
1058  }
1058
1059  @Override
1060  protected void onSizeChanged(int w, int h, int oldw,
1061      int oldh) {
1062      super.onSizeChanged(w, h, oldw, oldh);
1063
1064      if (w != oldw || h != oldh) {
1065          updateTextSize();
1066      }
1067  }
1067
1068  @Override
1069  protected void onMeasure(int widthMeasureSpec,
1070      int heightMeasureSpec) {
1071      super.onMeasure(widthMeasureSpec, heightMeasureSpec);
1072
1073      int width = MeasureSpec.getSize(widthMeasureSpec);
1074      int height = MeasureSpec.getSize(heightMeasureSpec);
1075
1076      if (width != 0 & height != 0) {
1077          setMeasuredDimension(width, height);
1078      }
1079  }
1080
1081  @Override
1082  protected void onDraw(Canvas canvas) {
1083      super.onDraw(canvas);
1084
1085      if (text != null) {
1086          canvas.drawText(text, 0, 0, paint);
1087      }
1088  }
1089
1090  @Override
1091  protected void onFocusChanged(boolean gainFocus,
1092      int status, KeyEvent keyEvent) {
1093      super.onFocusChanged(gainFocus, status, keyEvent);
1094
1095      if (gainFocus) {
1096          requestFocus();
1097          setSelection(0);
1098      }
1099  }
1099
1100  @Override
1101  protected void onLayout(boolean changed, int left,
1102      int top, int right, int bottom) {
1103      super.onLayout(changed, left, top, right, bottom);
1104
1105      if (changed) {
1106          updateTextSize();
1107      }
1108  }
1108
1109  @Override
1110  protected void onSizeChanged(int w, int h, int oldw,
1111      int oldh) {
1112      super.onSizeChanged(w, h, oldw, oldh);
1113
1114      if (w != oldw || h != oldh) {
1115          updateTextSize();
1116      }
1117  }
1117
1118  @Override
1119  protected void onMeasure(int widthMeasureSpec,
1120      int heightMeasureSpec) {
1121      super.onMeasure(widthMeasureSpec, heightMeasureSpec);
1122
1123      int width = MeasureSpec.getSize(widthMeasureSpec);
1124      int height = MeasureSpec.getSize(heightMeasureSpec);
1125
1126      if (width != 0 & height != 0) {
1127          setMeasuredDimension(width, height);
1128      }
1129  }
1130
1131  @Override
1132  protected void onDraw(Canvas canvas) {
1133      super.onDraw(canvas);
1134
1135      if (text != null) {
1136          canvas.drawText(text, 0, 0, paint);
1137      }
1138  }
1139
1140  @Override
1141  protected void onFocusChanged(boolean gainFocus,
1142      int status, KeyEvent keyEvent) {
1143      super.onFocusChanged(gainFocus, status, keyEvent);
1144
1145      if (gainFocus) {
1146          requestFocus();
1147          setSelection(0);
1148      }
1149  }
1149
1150  @Override
1151  protected void onLayout(boolean changed, int left,
1152      int top, int right, int bottom) {
1153      super.onLayout(changed, left, top, right, bottom);
1154
1155      if (changed) {
1156          updateTextSize();
1157      }
1158  }
1158
1159  @Override
1160  protected void onSizeChanged(int w, int h, int oldw,
1161      int oldh) {
1162      super.onSizeChanged(w, h, oldw, oldh);
1163
1164      if (w != oldw || h != oldh) {
1165          updateTextSize();
1166      }
1167  }
1167
1168  @Override
1169  protected void onMeasure(int widthMeasureSpec,
1170      int heightMeasureSpec) {
1171      super.onMeasure(widthMeasureSpec, heightMeasureSpec);
1172
1173      int width = MeasureSpec.getSize(widthMeasureSpec);
1174      int height = MeasureSpec.getSize(heightMeasureSpec);
1175
1176      if (width != 0 & height != 0) {
1177          setMeasuredDimension(width, height);
1178      }
1179  }
1180
1181  @Override
1182  protected void onDraw(Canvas canvas) {
1183      super.onDraw(canvas);
1184
1185      if (text != null) {
1186          canvas.drawText(text, 0, 0, paint);
1187      }
1188  }
1189
1190  @Override
1191  protected void onFocusChanged(boolean gainFocus,
1192      int status, KeyEvent keyEvent) {
1193      super.onFocusChanged(gainFocus, status, keyEvent);
1194
1195      if (gainFocus) {
1196          requestFocus();
1197          setSelection(0);
1198      }
1199  }
1199
1200  @Override

```

```

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,
17             getAssets(), Constants.KNITTINGFONTPATH));
18
19         public LineNumberTextView(Context context,
20             AttributeSet attrs) {
21             super(context, attrs);
22             setTypeface(Typeface.createFromAsset(context,
23                 getAssets(), Constants.KNITTINGFONTPATH));
24
25         }
26
27         public void updateLineNumbers(int lineCount) {
28             String linesString = "1";
29             for(int i = 1; i < lines; i++) {
30                 linesString += "\n" + (i+1);
31             }
32             setText(linesString);
33         }
34         private int measureLineTextWidth() {
35             return (int) getPaint().measureText(lines + " ");
36         }
37         public int getExactWidth() {
38             return measureLineTextWidth() +
39                 getPaddingRight() + getPaddingLeft();
40         }
41     }
42
43 }
```

Listing B.16: 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
8      * in.
9      */
10    private UUID id;
11    protected String name = "Default name";
12
13    public Metadata() {
14        id = UUID.randomUUID();
15
16    public String getFilename() {
17        return id + ".json";
18    }
19
20    public String getId() {
21        return id.toString();
22    }
23
24 }
```

```

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
8 public class Pattern extends Metadata {
9
10     private String[] patternRows = Constants.
11         DEFAULTPATTERNROWS;
12     private int rows = Constants.DEFAULTROWS;
13     private int columns = Constants.DEFAULTCOLUMNS;
14
15     @Override
16     public int compareTo(Metadata that) {
17         return this.name.compareTo(that.name);
18     }
19
20 }
```

Listing B.17: Metadata.java

```

1 package de.muffinworks.knittingapp.views;
2
3 import android.util.AttributeSet;
4
5 public class LineNumberTextView extends TextView {
6
7     private int linesCount;
8     String linesString = "1";
9
10    for(int i = 1; i < lines; i++) {
11        linesString += "\n" + (i+1);
12    }
13    setText(linesString);
14
15    private int measureLineTextWidth() {
16        return (int) getPaint().measureText(lines + " ");
17    }
18
19    public int getExactWidth() {
20        return measureLineTextWidth() +
21            getPaddingRight() + getPaddingLeft();
22    }
23
24 }
```

```

1 package de.muffinworks.knittingapp.views;
2
3 import android.util.AttributeSet;
4
5 public class LineNumberTextView extends TextView {
6
7     private int linesCount;
8     String linesString = "1";
9
10    for(int i = 1; i < lines; i++) {
11        linesString += "\n" + (i+1);
12    }
13    setText(linesString);
14
15    private int measureLineTextWidth() {
16        return (int) getPaint().measureText(lines + " ");
17    }
18
19    public int getExactWidth() {
20        return measureLineTextWidth() +
21            getPaddingRight() + getPaddingLeft();
22    }
23
24 }
```

Listing B.16: LineNumberTextView.java

```
14     private int currentRow = 1;
15
16     public Pattern() {
17         super();
18     }
19
20     public String[] getPatternRows() {
21         return patternRows;
22     }
23
24     public void setPatternRows(String[] patternRows) {
25         this.patternRows = patternRows;
26         this.rows = patternRows.length;
27         this.columns = PatternParser.parseRowToGridFormat(patternRows[0]).length;
28     }
29
30     public int getRows() {
31         return rows;
32     }
33
34     public int getColumnCount() {
35         return columns;
36     }
37
38     public int getCurrentRow() {
39         return currentRow;
40     }
41
42     private int currentRow = 1;
43
44     public void setCurrentRow(int currentRow) {
45         this.currentRow = currentRow;
46     }
47
48     @Override
49     public boolean equals(Object o) {
50         if (this == o) return true;
51         if (o == null || getClass() != o.getClass()) return false;
52         Pattern pattern = (Pattern) o;
53         if (rows != pattern.rows &&
54             columns != pattern.columns &&
55             currentRow != pattern.currentRow &&
56             Arrays.equals(patternRows, pattern.patternRows) &&
57             name.equals(pattern.name)) return true;
58     }
59
60     @Override
61     public int hashCode() {
62         Objects.hash(patternRows, rows, columns,
63                     currentRow, name);
64     }
65 }
```

Listing B.18: Pattern.java

```

50     new DialogInterface.OnClickListener {
51         @Override
52         public void onClick(DialogInterface dialog, int which) {
53             dialog.cancel();
54         }
55     }.create();
56 }
57 @Override
58 public void onAttach(Context context) {
59     super.onAttach(context);
60     if (context instanceof OnPatternDeleteInteractionListener) {
61         mClickListener = (OnPatternDeleteInteractionListener)
62             context;
63     } else {
64         throw new RuntimeException(context.toString());
65     }
66 }
67 }
68 }
69 }
70 @Override
71 public void onDetach() {
72     super.onDetach();
73     mListener = null;
74 }
75 public interface OnPatternDeleteInteractionListener {
76     void onConfirmDelete();
77 }
78 }
79 }



---



```

1 package de.muffinworks.knittingapp.views;
2 import android.content.Context;
3 import android.graphics.Canvas;
4 import android.graphics.Color;
5 import android.graphics.Paint;
6 import android.graphics.PointF;
7 import android.graphics.RectF;
8 import android.graphics.Typeface;
9 import android.util.AttributeSet;
10 import android.view.GestureDetector;
11 import android.view.MotionEvent;
12 import android.view.ScaleGestureDetector;
13 import android.view.View;
14 import android.view.ViewGroup;
15 import de.muffinworks.knittingapp.R;
16 import de.muffinworks.knittingapp.util.Constants;
17 import de.muffinworks.knittingapp.util.PatternParser;
18 import de.muffinworks.knittingapp.util.PatternParser;
19 /**
20 * Based on the tutorial on scroll gesture , dragging ,
21 * and scaling and the example project
22 * Interactivechart
23 * by Google's Android Developers found at
24 * https://developer.android.com/training/gestures/scroll.html
25 * The project's code is provided under the Apache
26 * License , Version 2.0
27 */
28 public class PatternGridView extends View {
29 private static final String TAG = "GridEditorView";
30 private boolean canBeEdited = true;
31 private final float CELLWIDTH = 100.0f;
32 private final float MARGIN = 40.0f;
33 private final float mCurrentRow = 0;
34 }

```

35 private final float ZOOMFACTORMIN = 0.5f;
36 private final float ZOOMFACTORMAX = 2.0f;
37 private final float DEFAULTSYMBOLTEXTSIZE = 60.0f;
38 private int rows = Constants.DEFAULTROWS;
39 private int columns = Constants.DEFAULTCOLUMNS;
40 private String[][] symbols = new String[columns][rows];
41 /**
42 * represents the grid content
43 */
44 /**
45 * represents the visible area on the screen minus
46 * the padding
47 */
48 /**
49 * represents the grid content
50 */
51 /**
52 * RowHighlightRect = new RectF();
53 */
54 private Paint mRowHighlightPaint;
55 private Paint mGridPaint;
56 private Paint mLabelTextPaint;
57 private float mScaleFactor = 1f;
58 private ScaleGestureDetector mScaleGestureDetector;
59 private GestureDetector mGestureDetector;
60 private PointF mTranslationOffset = new PointF(0, 0);
61 private boolean hasScrolled = false;
62 /**
63 */
64 private String mSelectedSymbol = null;
65 private int mCurrentRow;
66 /**
67 */
68 public PatternGridView(Context context) {
69 
```


```


```

Listing B.19: PatternDeleteDialogFragment.java

```

70     super(context);
71     init(context);
72   }
73   public PatternGridView(Context context, AttributeSet attrs) {
74     super(context, attrs);
75     init(context);
76   }
77   public PatternGridView(Context context, AttributeSet attrs, int defStyleAttr) {
78     super(context, attrs, defStyleAttr);
79     init(context, attrs, defStyleAttr);
80   }
81   private void init(Context context) {
82     initPaints();
83     updateContentRect();
84     mScaleGestureDetector = new ScaleGestureDetector(context,
85       (context, new GridScaleListener()));
86     mGestureDetector = new GestureDetector(context,
87       new GridGestureListener());
88   }
89   private void updateContentRect() {
90     mContentRect.set(
91       MARGIN,
92       MARGIN + columns * CELL_WIDTH *
93         mScaleFactor,
94       MARGIN + rows * CELL_WIDTH *
95         mScaleFactor
96     );
97   }
98   private void initPaints() {
99     mRowHighlightPaint = new Paint();
100    mRowHighlightPaint.setStrokeWidth(1);
101    mRowHighlightPaint.setColor(getResources().getColor(R.color.highlight_current_row));
102    mRowHighlightPaint.setAntiAlias(true);
103    mRowHighlightPaint.setStyle(Paint.Style.FILL);
104    mGridPaint = new Paint();
105    mGridPaint.setStrokeWidth(1);
106    mGridPaint.setColor(Color.BLACK);
107    mGridPaint.setStyle(Paint.Style.STROKE);
108    mLLabelTextPaint = new Paint();
109    mLLabelTextPaint.setAntiAlias(true);
110    mLLabelTextPaint.setTextAlign(Paint.Align.LEFT);
111    mLLabelTextPaint.setTextSize(20);
112    mLLabelTextPaint.setColor(Color.BLACK);
113    mSymbolPaint = new Paint();
114    mSymbolPaint.setAntiAlias(true);
115    mSymbolPaint.setTextAlign(Paint.Align.CENTER);
116    mSymbolPaint.setTextSize(DEFAULT_SYMBOL_TEXTSIZE);
117    mSymbolPaint.setAntiAlias(true);
118    mSymbolPaint.setTextAlign(Paint.Align.CENTER);
119    mSymbolPaint.setTextSize(DEFAULT_SYMBOL_SIZE);
120  }
121  Typeface knittingFont = Typeface.createFromAsset(
122    (getContext().getAssets(), Constants.KNITTING_FONT_PATH);
123    mSymbolPaint.setTypeface(knittingFont);
124  }
125  @Override
126  protected void onSizeChanged(int w, int h, int oldw,
127    int oldh) {
128    super.onSizeChanged(w, h, oldw, oldh);
129    mCanvasRect.set(
130      getPaddingLeft(),
131      getPaddingTop(),
132      getPaddingLeft() + w,
133      getPaddingTop() + h
134    );
135    if (mCurrentRow != 0) {
136      scrollCurrentRowToCenter();
137    }
138  }
139  public void scrollCurrentRowToCenter() {
140    float currentRowTop = getPixelPositionTopForRow(
141      mCurrentRow);
142    mTranslationOffset.x = mCanvasRect.height() / 2 - currentRowTop +
143      CELL_WIDTH;
144    clampOffset();
145    invalidate();
146  }
147  public int getRows() {
148    return rows;
149  }
150  public void setGridSize(int columns, int rows) {
151    this.rows = rows;
152    this.columns = columns;
153    String[][] newSymbols = new String[columns][rows];
154    this.rows = rows;
155    this.columns = columns;
156    newSymbols = new String[columns][rows];
157    // fill new array with data from old: data should
158    // persist in location, if new array is
159    // smaller than old, the data will be cut off and lost
160    if (rows > 0 && columns > 0) {
161      for (int c = 0; c < columns; c++) {
162        for (int r = 0; r < rows; r++) {
163          if (c < symbols.length && r <
164            symbols[0].length) {
165            newSymbols[c][r] = symbols[c][r];
166          } else {
167            newSymbols[c][r] = Constants.EMPTY_SYMBOL;
168          }
169        }
170      }
171    }
172    symbols = newSymbols;
173    updateContentRect();
174  }

```

```

173     invalidate();
174 }
175     public void setSymbol( int column , int row ) {
176         if (row >= 0 && row < rows && column >= 0 && column < columns) {
177             symbols [column] [row] = mSelectedSymbol;
178         }
179     }
180     public String [] setPattern( String [] patternRows ) {
181         String [] pattern = PatternParser .
182             parsePojoToGridFormat( patternRows );
183         setGridSize( pattern .length , pattern [0] .length );
184         symbols = pattern ;
185         invalidate();
186     }
187     public String [] getPattern() {
188         return PatternParser .parseGridFormatToPojo(
189             symbols );
190     }
191     public void setDeleteActive() {
192         mSelectedSymbol = Constants .EMPTY_SYMBOL;
193     }
194     public void setSelectedSymbol( String key ) {
195         mSelectedSymbol = key;
196     }
197     public void setSelectedKey( String key ) {
198         mSelectedSymbol = key;
199     }
200     public void setCurrentRow( int newCurrentRow ) {
201         mCurrentRow = newCurrentRow;
202         if (mCanvasRect .height () != 0.0 && mCurrentRow >
203             0)
204             scrollCurrentRowToCenter();
205     }
206     public void setCanBeEdited( boolean editable ) {
207         canBeEdited = editable;
208     }
209 }
210     @Override
211     public boolean onTouchEvent( MotionEvent event ) {
212         // see https://stackoverflow.com/questions-
213         // handle simple click tap
214         if (event .getAction () == MotionEvent .ACTION_UP
215             && canBeEdited) {
216             if (mSelectedSymbol != null && !hasScrolled )
217                 /> see 9965695 / how-to-distinguish -between -move -and
218                 -click -in -ontouchevent
219                 if (mSelectedSymbol != null && !hasScrolled )
220                     if (mSelectedSymbol != null && !hasScrolled )
221                         float x = event .getX ();
222                         float y = event .getY ();
223                         int row = calculateRowFromValue( y );
224                         int column = calculateColumnFromValue( x );
225
226
227         setSymbol( column , row );
228         postInvalidate();
229     } else {
230         hasScrolled = false;
231     }
232     return true;
233 }
234     boolean refVal = mScaleGestureDetector .
235         onTouchEvent( event );
236     refVal = mGestureDetector .onTouchEvent( event ) ||
237         refVal;
238     return refVal || super .onTouchEvent( event );
239 }
240     private int calculateRowFromValue( float y ) {
241         return (int )(y - mTranslationOffset .y - MARGIN
242             ) / (CELL_WIDTH * mScaleFactor );
243
244     private int calculateColumnFromValue( float x ) {
245         return (int )(x - mTranslationOffset .x - MARGIN
246             ) / (CELL_WIDTH * mScaleFactor );
247
248     private PointF getCellCenter( int column , int row ) {
249         return new PointF(
250             column * CELL_WIDTH *
251             MARGIN + column * CELL_WIDTH *
252             mScaleFactor ,
253             MARGIN + CELL_WIDTH / 2 * mScaleFactor ,
254             row * CELL_WIDTH *
255             mScaleFactor /
256             CELL_WIDTH / 2 * mScaleFactor +
257             ((int )Math .abs (mSymbolPaint .
258             getFontMetrics () .top )) / 2
259
260     private float getPixelPositionTopForRow( int row ) {
261         return mContentRect .top + (row - 1) * CELL_WIDTH *
262             * mScaleFactor;
263
264     private float getPixelPositionBottomForRow( int row ) {
265         return mContentRect .top + row * CELL_WIDTH *
266             mScaleFactor;
267
268     @Override
269     protected void onDraw( Canvas canvas ) {
270         super .onDraw( canvas );
271         canvas .save();
272         canvas .translate( mTranslationOffset .x ,
273             mTranslationOffset .y );
274         if (mCurrentRow != 0) {
275             mRowHighlightRect .set(
276                 mContentRect .left ,
277                 getPixelPositionTopForRow(
```

```

278     mCurrentRow,
279     mContentRect.left + columns *
280     CELL_WIDTH * mScaleFactor,
281     getPixelPositionBottomForRow(
282     mCurrentRow
283     );
284     canvas.drawRect(mRowHighlightRect,
285     mRowHighlightPaint);
286 }
287 canvas.restore();
288 }
289 private void drawSymbols(Canvas canvas) {
290     for(int c = 0; c < columns; c++) {
291         for(int r = 0; r < rows; r++) {
292             String symbol = symbols[c][r];
293             if (symbol != null) {
294                 mSymbolPaint.setTextSize(
295                     DEFAULT_SYMBOL_TEXTSIZE *
296                     mScaleFactor);
297                 PointF location = getCellCenter(c, r
298                     );
299                 canvas.drawText(
300                     symbol,
301                     location.x,
302                     location.y,
303                     mSymbolPaint
304                     );
305                 }
306             }
307         }
308     private void drawGrid(Canvas canvas) {
309         if (rows > 0 && columns > 0) {
310             for(int i = 0; i < rows + 1; i++) {
311                 canvas.drawLine(
312                     (i * CELL_WIDTH * mScaleFactor)
313                     + MARGIN,
314                     (columns * CELL_WIDTH *
315                         mScaleFactor) + MARGIN,
316                     (i * CELL_WIDTH * mScaleFactor)
317                     + MARGIN,
318                     (j * CELL_WIDTH * mScaleFactor)
319                     + MARGIN,
320                     canvas.drawLine(
321                     (j * CELL_WIDTH * mScaleFactor)
322                     + MARGIN,
323                     (j * CELL_WIDTH * mScaleFactor)
324                     + (rows * CELL_WIDTH *
325                         mScaleFactor) + MARGIN,
326                     5f - mTranslationOffset.x, // text
327                     width + offset
328                 );
329             }
330             private void drawAxisLabels(Canvas canvas) {
331                 // draw column labels
332                 for(int r = 0; r < rows; r++) {
333                     int text = r + 1;
334                     canvas.drawText(
335                         text + "n",
336                         5f - mTranslationOffset.x, // text
337                         r * CELL_WIDTH *
338                         mScaleFactor
339                         + CELL_WIDTH / 2 *
340                             ((int) Math.abs(
341                                 (mLabelTextPaint.getFontMetrics()
342                                     .top) / 2
343                                     + MARGIN
344                                     )
345                                     + mLabelTextPaint
346                                     );
347                 }
348             }
349             canvas.drawText(
350                 text + "n",
351                 c * CELL_WIDTH *
352                     mScaleFactor
353                     + MARGIN
354                     + mLabelTextPaint
355                     - mLabelTextPaint
356                     + "n" ) / 2
357             );
358         }
359     }
360     public void resetZoom() {
361         mScaleFactor = 1.0f;
362         invalidate();
363     }
364 }
365 class GestureDetector extends GestureDetector {
366     SimpleOnGestureListener {
367         @Override
368         public boolean onScroll(MotionEvent e1,
369             MotionEvent e2, float distanceX,
370             distanceY) {
371             // minus operation because scroll is inverse
372             // to dragging
373             mTranslationOffset.x -= distanceX;
374             mTranslationOffset.y -= distanceY;
375         }
376     }
377 }

```

```

373     clampOffset();
374     hasScrolled = true;
375     postInvalidate();
376     return true;
377   }
378
379   private void clampOffset() {
380     float maxRightOffset = mCanvasRect.width() -
381       mContentRect.width() - 2 * MARGIN;
382     float maxDownOffset = mCanvasRect.height() -
383       mContentRect.height() - 2 * MARGIN;
384     if (mTranslationOffset.x > 0.0f || 
385       mTranslationOffset.x > 0) {
386       if (mTranslationOffset.y > 0.0f || 
387         mTranslationOffset.y > 0) {
388         mTranslationOffset.x = 0.0f;
389         mTranslationOffset.y = 0.0f;
390       } else {
391         mTranslationOffset.y = 0.0f;
392         if (maxRightOffset < 0 && mTranslationOffset.x <
393           maxRightOffset) {
394           mTranslationOffset.x = maxRightOffset;
395         } else {
396           if (maxDownOffset < 0 && mTranslationOffset.y <
397             maxDownOffset) {
398             mTranslationOffset.y = maxDownOffset;
399           }
400         }
401       class GridScaleListener extends ScaleGestureDetector
402         .SimpleOnScaleGestureListener {
403         private PointF viewportFocus = new PointF();
404         @Override
405         public boolean onScale(ScaleGestureDetector
406           detector) {
407           mScaleFactor *= detector.getScaleFactor();
408           mScaleFactor = Math.min(
409             mScaleFactor, ZOOMFACTORMAX),
410           updateContentRect();
411           viewportFocus.set(
412             detector.getFocusX(),
413             detector.getFocusY());
414           invalidate();
415           return true;
416         }
417       }
418     }
419   }

```

Listing B.20: PatternGridView.java

```

1 package de.muffinworks.knittingapp;
2 import android.content.DialogInterface;
3 import android.os.Bundle;
4 import android.support.annotation.Nullable;
5 import android.support.v4.app.FragmentManager;
6 import android.support.v4.view.Menu;
7 import android.support.v4.widget.FloatingActionButton;
8 import android.support.v4.widget.FragmentManager;
9 import android.view.Menu;
10 import android.view.View;
11 import android.view.ViewGroup;
12 import android.widget.ListView;
13 import com.google.gson.JsonSyntaxException;
14 import java.io.IOException;
15 import java.util.List;
16 import java.util.ListAdapter;
17 import de.muffinworks.knittingapp.fragments
18   .PatternNameDialogFragment;
19 import de.muffinworks.knittingapp.storage.models.Pattern
20 import de.muffinworks.knittingapp.util.Constants;
21 import de.muffinworks.knittingapp.views.Adapters;
22 import de.muffinworks.knittingapp.views.OnClickListerner
23 implements PatternNameDialogFragment.
24 OnPatternNameInteractionListener {
25
26   private ListView mPatternsList;
27   private PatternListAdapter mAdapter;
28   private FloatingActionButton mFab;
29   private MenuItem mExportAllMenu = null;
30
31   @Override
32   protected void onCreate(@Nullable Bundle
33     savedInstanceState) {
34     super.onCreate(savedInstanceState);
35     setContentView(R.layout.activity_pattern_list);
36     enableActionBar(false);
37
38     mPatternsList = (ListView) findViewById(R.id.
39       patterns_list);
40     mAdapter = new PatternListAdapter(this);
41     mPatternsList.setAdapter(mAdapter);
42     mPatternsList.setItemsCanFocus(true);
43
44     mFab = (FloatingActionButton) findViewById(R.id.
45       fab);
46     mFab.setOnClickListener(new View.OnClickListener {
47       @Override
48       public void onClick(View v) {
49         showSetNameDialog();
50     }
51   }

```

```

50    });
51    requestExternalStoragePermission();
52  }
53  @Override
54  protected void onResume() {
55    super.onResume();
56    mAdapter.notifyDataSetChanged();
57    checkExportAvailability();
58  }
59 }
60
61  @Override
62  public boolean onCreateOptionsMenu(Menu menu) {
63    getMenuInflater().inflate(R.menu.menu,
64    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  {
79    int id = item.getItemId();
80    if (id == R.id.import_pattern) {
81      importFile();
82    } else {
83      requestExternalStoragePermission();
84    } else if (id == R.id.export_all) {
85      if (isExternalStoragePermissionGranted()) {
86        exportAllPatterns();
87      } else {
88        requestExternalStoragePermission();
89      }
90    }
91    return super.onOptionsItemSelected(item);
92  }
93
94  private void exportAllPatterns() {
95    try {
96      mStorage.exportAll();
97      showAlertDialog(getString(R.string.success_export_all,
98      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.FILEPICKER_REQUEST_CODE);
108  }
109
110  @Override
111  protected void onActivityResult(int requestCode, int
112    resultCode, Intent data) {
113    if (requestCode == Constants.FILEPICKER_REQUEST_CODE) {
114      if (data != null) {
115        try {
116          Pattern importedPattern =
117            mStorage.loadFromFile(data.getData());
118        } catch (FileNotFoundException e) {
119          showAlertDialog(getString(R.string.info_import_pattern_already_exists));
120        }
121      }
122    }
123  }
124
125  mStorage.save(importedPattern);
126
127  } catch (JsonSyntaxException e) {
128    showAlertDialog(getString(R.string.error_import_no_json));
129  }
130
131  }
132
133  }
134
135  FragmentManager fm = getSupportFragmentManager();
136  PatternNameDialogFragment dialog =
137    dialog.show(fm, getString(R.string.tag_dialog_fragment_set_name));
138
139  @Override
140  public void onSetName(String name) {
141    Pattern pattern = new Pattern();
142    pattern.setName(name);
143    String patternId = pattern.getId();
144

```

Listing B.21: PatternListActivity.java

```

86     OnClickListener() {
87         @Override
88         public void onClick(View v) {
89             String patternId = ((Metadata) getItem(
90                 position)).getId();
91             Intent intent = new Intent(mContext,
92                 ViewerActivity.class);
93             intent.putExtra(Constants.EXTRA_PATTERNSID,
94                 patternId);
95             mContext.startActivity(intent);
96         }
97         return convertView;
98     }
99     public void confirmDeleteDialog(final String id,
100         String name) {
101         AlertDialog dialog = new AlertDialog.Builder(
102             mContext)
103             .setTitle(mContext.getString(R.string
104                 dialog_title_pattern_delete, name))
105             .setPositiveButton(R.string.dialog_ok,
106                 new DialogInterface.OnClickListener() {
107                     @Override
108                     public void onClick(DialogInterface dialog,
109                         int which) {
110                         PatternStorage storage =
111                             PatternStorage.getInstance();
112                         storage.delete(id);
113                         notifyDataSetChanged();
114                     }
115                 });
116         dialog.create();
117         dialog.show();
118     }
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.22: PatternListAdapter.java

```

```
1 package de.muffinworks.knittingapp.fragments;
2 import android.app.Dialog;
3 import android.content.Context;
4 import android.content.DialogInterface;
5 import android.os.Bundle;
6 import android.support.annotation.NonNull;
7 import android.support.annotation.Nullable;
8 import android.support.v4.app.DialogFragment;
9 import android.support.v7.app.AlertDialog;
10 import android.text.Editable;
11 import android.text.InputFilter;
12 import android.text.TextSpanned;
13 import android.text.TextWatcher;
14 import android.widget.EditText;
15 import android.widget.LinearLayout;
16 import java.util.regex.Pattern;
17 import de.muffinworks.knittingapp.R;
18 import de.muffinworks.knittingapp.util.Constants;
19
20 public class PatternNameDialogFragment extends DialogFragment {
21
22     private static final String BUNDLE_NAME = "name";
23
24     @Override
25     protected void onCreate(@Nullable Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         if (getArguments() != null) {
28             mName = getArguments().getString(BUNDLE_NAME);
29         }
30     }
31
32     @Override
33     public void onSaveInstanceState(Bundle savedInstanceState) {
34         savedInstanceState.putString(BUNDLE_NAME, mName);
35     }
36
37     @Override
38     public void onDestroy() {
39         super.onDestroy();
40         if (mName != null) {
41             Intent intent = new Intent();
42             intent.putExtra("name", mName);
43             setResult(RESULT_OK, intent);
44             dismiss();
45         }
46     }
47
48     @Override
49     public void onDismiss(DialogInterface dialog) {
50         super.onDismiss(dialog);
51         if (mName != null) {
52             Constants.setPatternName(mName);
53         }
54     }
55 }
```

Listing B.23: PatternNameDialogFragment.java

```
1 package de.muffinworks.knittingapp.util;
2 import java.util.ArrayList;
3 import java.util.Arrays;
4 import java.util.List;
```



```

102    to get number of columns
103    int symbolCount = 0;
104    String expandedRow = "";
105    for (String group : groupedSymbols) {
106        //remove all letters
107        String symbolFactor = group.replaceAll(
108            REGEX_ALL_NONDIGITS_END, EMPTY);
109        //remove all numbers
110        String symbol = group.replaceAll(
111            REGEX_ALL_DIGITS, EMPTY);
112        if (!symbolFactor.isEmpty()) {
113            //was grouped symbol e.g. 33k
114            int factor = Integer.parseInt(
115                symbolFactor);
116            if ((symbolCount + factor) >
117                Constants.MAX_ROWS_AND_COLUMNS_LIMIT) {
118                symbolCount += factor;
119                for (int j = 0; j < factor; j++) {
120                    expandedRow += symbol;
121                }
122            } else {
123                //only one symbol, not grouped e.g.
124                int k;
125                symbolCount++;
126                expandedRow += symbol;
127            }
128        }
129        if (symbolCount > columns) columns =
130            symbolCount;
131        expandedRows.add(expandedRow);
132    }
133    String[][] result = new String[columns][
134        compressedRows.length];
135    // iterate over String [][] and fill in from row
136    strings
137    for (int r = 0; r < compressedRows.length; r++) {
138        for (int c = 0; c < expandedRows.get(r).length();
139            length(); c++) {
140            result[c][r] = Character.toString(
141                expandedRows.get(r).charAt(c));
142        }
143    }
144    // fill null places with placeholder for empty
145    cell
146    to get number of columns
147    int symbolCount = 0;
148    String[] strings = result[c];
149    for (String group : groupedSymbols) {
150        //remove all letters
151        String symbolFactor = group.replaceAll(
152            REGEX_ALL_NONDIGITS_END, EMPTY);
153        //remove all numbers
154        String symbol = group.replaceAll(
155            REGEX_ALL_DIGITS, EMPTY);
156        if (!symbolFactor.isEmpty()) {
157            //was grouped symbol e.g. 33k
158            int factor = Integer.parseInt(
159                symbolFactor);
160            if ((symbolCount + factor) >
161                Constants.MAX_ROWS_AND_COLUMNS_LIMIT) {
162                symbolCount += factor;
163                expandedRows.add(expandedRow);
164            }
165            String[] rows = rowInput.split(
166                REGEX_LOOKBEHIND_LINEFEED);
167            int symbolsPerRow = new int[rows.length];
168            int columns = 1;
169            for (int r = 0; r < rows.length; r++) {
170                ArrayList<MatchResult> groupedSymbols = new
171                ArrayList();
172                rows[r].replaceAll(
173                    REGEX_ALL_DIGITS_END, EMPTY);
174                rows[r] = rows[r].replaceAll(LINEFEED, EMPTY);
175                Matcher m = pattern.matcher(rows[r]);
176                while (m.find()) {
177                    groupedSymbols.add(m.toMatchResult());
178                }
179            }
180            int columnCount = 0;
181            for (MatchResult group : groupedSymbols) {
182                String symbolFactor = group.group(1);
183                String symbol = group.group(2);
184                if (columnCount >= Constants.MAX_ROWS_AND_COLUMNS_LIMIT)
185                    break;
186            }
187            if (!symbolFactor.isEmpty()) {
188                int factor = Integer.parseInt(
189                    symbolFactor);
190                if (factor + columnCount > Constants.MAXROWS_AND_COLUMNSLIMIT) {
191                    factor = Constants.MAXROWS_AND_COLUMNSLIMIT -
192                        columnCount;
193                }
194            }
195        }
196    }
197    String[] result = result[c];
198    for (int r = 0; r < result.length; r++) {
199        for (int c = 0; c < result[r].length(); c++) {
200            result[r][c] = result[c][r];
201        }
202    }
203    return result;
204}

```



```

216     parseGridToRowFormat(in);
217     assertTrue(expected.equals(result));
218 }
219
220 @Test
221 public void convertToString_3x2WithEmptySpaces() {
222     String[] in = {"", "", "", "", ""};
223     String[] expected = {"h", "f", "g", "j", "h", "g", "j"};
224     ArrayList<String> result = new ArrayList<>();
225     result.add("3h");
226     result.add("f2g");
227     result.add("2.j");
228     assertEquals(result.length == expected.size());
229     for (int i = 0; i < result.length; i++) {
230         if (!result[i].equals(expected.get(i))) fail();
231     }
232 }
233
234 @Test
235 public void pojoToGrid() {
236     String[] expected = {"", "", "", "", ""};
237     String[] result = {"h", "g", "j", "h", "g", "j"};
238     String[] in = {"", "", "", "", ""};
239     String[] result = {"3h", "f2g", "2.j", "3h", "f2g", "2.j"};
240     String[] expected = {"2n\\n2g\\n2h"};
241     assertEquals(result == PatternParser.parseGridFormat(in));
242     assertEquals(result == Arrays.deepEquals(result, expected));
243     assertEquals(result == PatternParser.parsePojoToGridFormat(in));
244     assertEquals(result == PatternParser.parsePojoToRowFormat(in));
245     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
246     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
247     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
248     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
249     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
250     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
251     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
252     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
253     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
254     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
255     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
256     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
257     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
258     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
259     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
260     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
261     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
262     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
263     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
264     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
265     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
266     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
267     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
268     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
269     assertEquals(result == PatternParser.parseRowFormatToPojo(result));
270 }

271
272 @Test
273 public void gridToPojo() {
274

```

```

72     @Test
73     public void rowToGridWithTrailingNumber() {
74         String in = "3r2";
75         String[] expected = {
76             {"3", "r", "2", "\n"},
77             {"3", "r", "2", "\n"}
78         };
79         String[] result = PatternParser.
80             parseRowToGridFormat(in);
81         assertEquals(Arrays.deepEquals(result, expected));
82     }
83     @Test
84     public void rowToGridEmptyFirstRow() {
85         String in = "\n3h";
86         String[] expected = {
87             "3",
88             "3h"
89         };
90         String[] result = PatternParser.
91             parseRowFormatToPojo(in);
92         assertEquals(Arrays.deepEquals(result, expected));
93     }
94     @Test
95     public void rowToGridEmptyRowInMiddle() {
96         String in = "4f\n3h";
97         String[] expected = {
98             "4f",
99             "3h"
100        };
101        String[] result = PatternParser.
102            parseRowFormatToPojo(in);
103        assertEquals(Arrays.deepEquals(result, expected));
104    }
105    @Test
106    public void rowToGridEmptyRowAtEnd() {
107        String in = "4f\n3h\n";
108        String[] expected = {
109            "4f", "3h", "\n",
110            "4f", "3h", "\n",
111            "4f", "3h", "\n",
112            "4f", "3h", "\n"
113        };
114        String[] result = PatternParser.
115            parseRowToGridFormat(in);
116        assertEquals(Arrays.deepEquals(result, expected));
117    }
118    @Test
119    public void rowToPojoEmptyRowAtEnd() {
120        String in = "\n4f\n3h\n";
121        String[] expected = {
122            "4f",
123            "4f",
124            "4f",
125            "3h"
126        };
127        String[] result = PatternParser.
128            parseRowFormatToPojo(in);
129        assertEquals(Arrays.deepEquals(result, expected));
130    }
131    @Test
132    public void rowToGridMoreThanMaxColumns() {
133        String in = "5553d";
134        String[] expected = {
135            "5553d", "\n",
136            "MAX_ROWSANDCOLUMNS_LIMIT + " + "h";
137        };
138        String[] result = PatternParser.
139            parseRowFormatToPojo(in);
140        assertEquals(Arrays.deepEquals(result, expected));
141    }
142    @Test
143    public void rowToGridMoreThanMaxColumns() {
144        String in = "555g";
145        String[] expected = {
146            "555g", "\n",
147            "MAX_ROWSANDCOLUMNS_LIMIT + " + "h";
148        };
149        String[] result = PatternParser.
150            parseRowFormatToPojo(in);
151        assertEquals(Arrays.deepEquals(result, expected));
152    }
153    @Test
154    public void rowToGridFormat() {
155        String in = "4f\n3h\n";
156        String[] expected = {
157            "4f", "3h", "\n"
158        };
159        String[] result = PatternParser.
160            parseRowToGridFormat(in);
161        assertEquals(Arrays.deepEquals(result, expected));
162    }
163    @Test
164    public void rowToGridFormat() {
165        String in = "4f\n3h\n";
166        String[] expected = {
167            "4f", "3h", "\n"
168        };
169        String[] result = PatternParser.
170            parseRowFormatToPojo(in);
171        assertEquals(Arrays.deepEquals(result, expected));
172    }
173    @Test
174    public void rowToGridEmptyRow() {
175        String in = "\n";
176        String[] expected = {
177            "\n"
178        };
179        String[] result = PatternParser.
180            parseRowFormatToPojo(in);
181        assertEquals(Arrays.deepEquals(result, expected));
182    }
183    @Test
184    public void rowToGridEmptyRow() {
185        String in = "\n";
186        String[] expected = {
187            "\n"
188        };
189        String[] result = PatternParser.
190            parseRowFormatToPojo(in);
191        assertEquals(Arrays.deepEquals(result, expected));
192    }
193    @Test
194    public void rowToGridEmptyRow() {
195        String in = "\n";
196        String[] expected = {
197            "\n"
198        };
199        String[] result = PatternParser.
200            parseRowFormatToPojo(in);
201        assertEquals(Arrays.deepEquals(result, expected));
202    }
203    @Test
204    public void rowToGridEmptyRow() {
205        String in = "\n";
206        String[] expected = {
207            "\n"
208        };
209        String[] result = PatternParser.
210            parseRowFormatToPojo(in);
211        assertEquals(Arrays.deepEquals(result, expected));
212    }
213    @Test
214    public void rowToGridEmptyRow() {
215        String in = "\n";
216        String[] expected = {
217            "\n"
218        };
219        String[] result = PatternParser.
220            parseRowFormatToPojo(in);
221        assertEquals(Arrays.deepEquals(result, expected));
222    }
223    @Test
224    public void rowToGridEmptyRow() {
225        String in = "\n";
226        String[] expected = {
227            "\n"
228        };
229        String[] result = PatternParser.
230            parseRowFormatToPojo(in);
231        assertEquals(Arrays.deepEquals(result, expected));
232    }
233    @Test
234    public void rowToGridEmptyRow() {
235        String in = "\n";
236        String[] expected = {
237            "\n"
238        };
239        String[] result = PatternParser.
240            parseRowFormatToPojo(in);
241        assertEquals(Arrays.deepEquals(result, expected));
242    }
243    @Test
244    public void rowToGridEmptyRow() {
245        String in = "\n";
246        String[] expected = {
247            "\n"
248        };
249        String[] result = PatternParser.
250            parseRowFormatToPojo(in);
251        assertEquals(Arrays.deepEquals(result, expected));
252    }
253    @Test
254    public void rowToGridEmptyRow() {
255        String in = "\n";
256        String[] expected = {
257            "\n"
258        };
259        String[] result = PatternParser.
260            parseRowFormatToPojo(in);
261        assertEquals(Arrays.deepEquals(result, expected));
262    }
263    @Test
264    public void rowToGridEmptyRow() {
265        String in = "\n";
266        String[] expected = {
267            "\n"
268        };
269        String[] result = PatternParser.
270            parseRowFormatToPojo(in);
271        assertEquals(Arrays.deepEquals(result, expected));
272    }
273    @Test
274    public void rowToGridEmptyRow() {
275        String in = "\n";
276        String[] expected = {
277            "\n"
278        };
279        String[] result = PatternParser.
280            parseRowFormatToPojo(in);
281        assertEquals(Arrays.deepEquals(result, expected));
282    }

```

 Listing B.25: 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 import java.io.File;
11 import java.io.FileInputStream;
12 import java.io.FileNotFoundException;
13 import java.io.FileOutputStream;
14 import java.io.FileReader;
15 import java.io.FileWriter;
16 import java.io.IOException;
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 import de.muffinworks.knittingapp.R;
24 import de.muffinworks.knittingapp.storage.models.
25 Metadata;
26 import de.muffinworks.knittingapp.storage.models.Pattern;
27 import de.muffinworks.knittingapp.util.Constants;
28 public class PatternStorage {
29
30     private static final String TAG = "PatternStorage";
31
32     private Context mContext;
33     private Gson mGson = new Gson();
34     private HashMap<String, Metadata> mDataTable;
35
36     private static PatternStorage storage = new
37     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;
51         loadMetadata();
52     }
53
54     private String getApplicationDir() {
55
56         return mContext.getFilesDir().getPath();
57
58         private String getFilePathInApplicationDir(String
59             fileName) {
60
61             return getApplicationDir() + "/" + fileName;
62
63             private boolean isExternalStorageWritable() {
64                 return Environment.MEDIA_MOUNTED.equals(
65                     Environment.getExternalStorageState()) &&
66                     Environment.getExternalStorageDirectory()
67                         .canWrite();
68
69             public void exportAll() throws IOException {
70                 for (String id : mDataTable.keySet()) {
71                     export(id);
72
73                     public File export(String id) throws IOException {
74                         if (!isExternalStorageWritable())
75                             throw new IOException(mContext.getString(R.
76                                 string.error_external_storage_not_mounted));
77
78                     File patternFile = new File(
79                         getFilePathInApplicationDir(id + ".json"));
79
80                     File file = new File(Environment.
81                         getExternalStorageDirectory());
82
83                     file.mkdirs();
84                     file = new File(file, id + ".json");
85                     copyFile(patternFile, file);
86                     return file;
87
88
89                     /**
90                      * From https://stackoverflow.com/questions/9292954/
91                      * how-to-make-a-copy-of-a-file-in-android
92
93                     private void copyFile(File src, File dst) throws
94                         IOException {
95                         FileOutputStream outStream = new
96                         FileOutputStream(dst);
97                         FileInputStream inStream = new FileInputStream(
98                             src);
99                         FileChannel inChannel = inStream.getChannel();
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
```

```

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(getApplicationContextDir(),
107                Constants.METADATAFILENAME);
108            FileReader fileReader = new FileReader(file)
109                // https://sites.google.com/site/gson/gson-
110                // user-guide#TOC-Collections_Examples-
111                Type metadataListType = new TypeToken<List<
112                Type metadata>().getGenericType();
113                List<Metadata> metadata = Gson.fromJson(
114                    fileReader, metadataListType);
115
116                fileReader.close();
117                catch (IOException e) {
118                    logError(getApplicationContext(), e);
119                    error_load_metadata());
120                }
121
122                try {
123                    if (metadata != null) {
124                        for (Metadata m : metadata) {
125                            mMetaDataTable.put(m.getId(), m);
126                        }
127                    }
128
129                    private void updateMetadata() {
130                        try {
131                            File file = new File(getApplicationContextDir(),
132                                Constants.METADATAFILENAME);
133                            String json = Gson.toJson(mMetaDataTable,
134                                values());
135                            FileWriter fileWriter = new FileWriter(file);
136                            fileWriter.write(json);
137                            catch (IOException e) {
138                                logError(getApplicationContext(), e);
139                                error_update_metadata());
140                            }
141
142                    public Metadata[] listMetadataEntries() {
143                        Metadata[] m = mMeteDataDataTable.values();
144                        new Metadata[mMeteDataDataTable.size()];
145
146                    if (m.length > 0) {
147                        Arrays.sort(m);
148                    }
149                    return m;
150
151                    public void save(Pattern pattern) {
152                        try {
153                            FileWriter fileWriter = new FileWriter(
154                                getFilePathInApplicationDir(pattern));
155                            fileWriter.write(mGson.toJson(pattern));
156                            fileWriter.close();
157                            // call clone to put only metadata
158                            // information into hashmap, not actual
159                            // pattern related
160                            // information -> needs less resources
161                            mMeteDataTable.put(pattern.getId(), pattern.
162                                clone());
163                            updateMetadata();
164                            catch (IOException e) {
165                                logError(getApplicationContext(), e);
166                                error_savePattern());
167                            }
168
169                            public Pattern loadFromFile(String path) throws
170                                JsonSyntaxException {
171                                try {
172                                    FileReader reader = new FileReader(path);
173                                    return mGson.fromJson(reader, Pattern.class)
174                                }
175                                catch (FileNotFoundException e) {
176                                    logError(getApplicationContext(), e);
177                                    return null;
178                                }
179
180                                public boolean checkPatternDuplicate(Pattern pattern
181                                ) {
182                                    String id = pattern.getId();
183                                    return mMeteDataTable.containsKey(id);
184
185                                public Pattern load(String id) throws
186                                    JsonSyntaxException {
187                                    return loadFromFile(getFilePathInApplicationDir(
188                                        id + ".json"));
189
190                                public void clearAll() {
191                                    for (Metadata m : mMeteDataTable.values()) {
192                                        getFileFromApplicationDir(m.getFilename())
193                                            .delete();
194
195                                }
196
197                                getFileFromApplicationDir(Constants.
198                                METADATAFILENAME).delete();
199
200                                mMeteDataTable.clear();
201
202                            }
```

```

191     }
192 
193     public void delete(Metadata pattern) {
194         getFileFromApplicationDir(pattern.getFilename())
195             .delete();
196         mMataDataTable.remove(pattern.getId());
197         updateMetadata();
198     }
199 
200     public void delete(String id) {
201         delete(mMataDataTable.get(id));
202     }

```

```

203     private File getFileFromApplicationDir(String
204         filename) {
205         return new File(getApplicationDir() , filename);
206     }
207 
208     private void logError(String message) {
209         Log.e(TAG, message);
210     }

```

Listing B.26: PatternStorage.java

```

1    package de.muffinworks.knittingapp.storage;
2 
3    import android.content.Context;
4    import android.os.Environment;
5    import android.support.test.InstrumentationRegistry;
6    import android.support.test.runner.AndroidJUnit4;
7 
8    import org.junit.After;
9    import org.junit.Before;
10   import org.junit.Test;
11   import org.junit.runner.RunWith;
12 
13   import java.io.File;
14   import java.io.IOException;
15 
16   import de.muffinworks.knittingapp.storage.models.Pattern;
17   import de.muffinworks.knittingapp.storage.models.PatternMetadata;
18   import static org.junit.Assert.assertEquals;
19   import static org.junit.Assert.assertNotNull;
20   import static org.junit.Assert.assertNull;
21   import static org.junit.Assert.assertTrue;
22   import static org.junit.Assert.fail;
23 
24   @RunWith(AndroidJUnit4.class)
25   public class PatternStorageTest {
26 
27     private Context context = InstrumentationRegistry
28         .getInstrumentation()
29         .getTargetContext();
30 
31     @Before
32     public void setup() throws IOException {
33         storage = PatternStorage.getInstance();
34         storage.init(context);
35         storage.clearAll();
36 
37         @Test
38         public void saveAndLoadPatternTest() throws
39             IOException {
40             assertNull(storage.load("fileIdThatDoesNotExist"));
41             Pattern pattern = new Pattern();
42 
43             pattern.setName("scarf");
44             pattern.setCurrentRow(2);
45             pattern.setPatternRows(new String[] {
46                 "3e",
47                 "2er",
48                 "ttf",
49                 "3t",
50             });
51             assertEquals(3, pattern.getColumns());
52             assertEquals(5, pattern.getRows());
53 
54             storage.save(pattern);
55             File test = new File(context.getFilesDir()
56                 .getPath() + "test");
57             assertTrue(test.exists());
58             assertEquals(pattern.getName());
59             Pattern p2 = storage.load(pattern.getId());
60             assertEquals(pattern.equals(p2), true);
61 
62         }
63 
64         @Test
65         public void listEntriesTest() throws IOException {
66             assertEquals(storage.listMetadataEntries().length
67                 == 0);
68             for(int i = 1; i <= 5; i++) {
69                 Pattern pattern = new Pattern();
70                 storage.save(pattern);
71                 assertTrue(storage.listMetadataEntries()
72                     .length == i);
73 
74             }
75 
76         }
77 
78         @Test
79         public void deleteTest() throws IOException {
80             saveAndLoadPatternTest();
81             PatternStorage storage2 = PatternStorage
82                 .getInstance();
83             assertEquals(storage2.listMetadataEntries().length
84                 > 0);
85             String id = storage2.listMetadataEntries()[0];
86             storage.delete(id);
87 
88         }
89 
90         @Test
91         public void deletePattern() throws
92             IOException {
93             storage.delete("fileIdThatDoesNotExist");
94             assertEquals(storage.listMetadataEntries().length
95                 == 0);
96         }
97 
98     }

```

Listing B.26: PatternStorage.java

```

82     for (Metadata m : storage2.listMetadataEntries()
83         ) {
84         if (m.getId() .equals (id) )
85             }
86     }
87     @Test
88     public void exportTest() throws IOException {
89         Pattern pattern = new Pattern ();
90         pattern.setName ("scarf");
91         pattern.setCurrentRow (2);
92         pattern.setPatternRows (new String []{
93             "\u003e",
94             "\u002er\u003e",
95             "\u002ett\u003e",
96             "\u003ff\u003e",
97             "\u003tt\u003e",
98         });
99         storage . save (pattern);
100        File file = storage . export (pattern.getId ());
101        assertTrue (file . exists ());
102    }
103    @Test
104    public void importTest() throws IOException {
105        Pattern pattern = new Pattern ();
106        pattern.setName ("scarf");
107        pattern.setCurrentRow (2);
108        pattern.setPatternRows (new String []{
109            "\u003e",
110            "\u002er\u003e",
111            "\u002ett\u003e",
112            "\u003ff\u003e",
113            "\u003tt\u003e",
114            "\u003t\u003e",
115        });
116        storage . save (pattern);
117        File file = storage . export (pattern.getId ());
118        assertTrue (file . exists ());
119    }
120    storage . delete (pattern.getId ());
121    storage . importPattern (file . getPath ());
122    Pattern pattern2 = storage . load (pattern.getId ());
123    assertEquals (pattern , pattern2);
124    assertEquals (pattern , pattern2);
125    }
126    @Test
127    public void exportAllTest() throws IOException {
128        storage . exportAll ();
129        for (Metadata md : storage . listMetadataEntries ()
130            ) {
131            File file = new File (Environment . getExternalStoragePublicDirectory (
132                Environment . DIRECTORY_DOCUMENTS) , md.getId () +
133                ".json");
134            assertTrue (file . exists ());
135        }
136    }
137    @After
138    public void cleanUp() throws IOException {
139        storage . clearAll ();
140    }
141}

```

Listing B.27: PatternStorageTest.java

```

1 package de.muffinworks.knittingapp.fragments;
2 import android.os.Bundle;
3 import android.support.annotation.Nullable;
4 import android.support.design.widget.Snackbar;
5 import android.support.v4.app.Fragment;
6 import android.view.LayoutInflater;
7 import android.view.View;
8 import android.view.ViewGroup;
9 import android.widget.GridView;
10 import android.widget.AdapterView;
11 import java.util.Arrays;
12 import java.util.List;
13 import de.muffinworks.knittingapp.R;
14 import de.muffinworks.knittingapp.layouts.*;
15 import RowEditorLinearLayout;
16 import de.muffinworks.knittingapp.storage.PatternStorage;
17 import de.muffinworks.knittingapp.storage.models.Pattern;
18 import de.muffinworks.knittingapp.views.adapters.*;
19 import KeyboardTypingAdapter;
20 public class RowEditorFragment extends Fragment {
21     private static final String TAG = "RowEditorFragment";
22     private RowEditorLinearLayout mRowEditorView;
23     private Pattern mPattern;
24     private PatternStorage mStorage;
25     public static RowEditorFragment getInstance (String patternId) {
26         RowEditorFragment fragment = new
27             RowEditorFragment();
28         if (patternId != null) {
29             Bundle bundle = new Bundle();
30             bundle.putString ("id", patternId);
31             fragment.setArguments (bundle);
32             return fragment;
33         }
34     }
35     @Override
36     public void onCreate (@Nullable Bundle savedInstanceState) {
37         super.onCreate(savedInstanceState);
38         savedInstanceState;
39     }

```

```

super.onCreate(savedInstanceState);
mStorage = PatternStorage.getInstance();
mStorage.init(getActivity());
if (getArguments().getBoolean("null")) {
    mPattern = mStorage.load getArguments();
    getString("id"));
}
}

@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 =
(findViewById(R.id.keyboard_gridview);
mKeyboard.setAdapter(new KeyboardTypingAdapter(
getActivity(), this));
view.findViewById(R.id.op-enter).
setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    onEnter();
}
});
view.findViewById(R.id.op-delete).
setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    onDelete();
}
});
}
}

@Override
public void onKeyClicked(String key) {
int start = mRowEditorView.getTextStart();
getSelectionStart();
mRowEditorView.getText().insert(
start, key.toUpperCase());
}

@Override
public void onTextChanged() {
mPattern.notifyDataChanged();
mPattern = mStorage.load(mPattern.getId());
mRowEditorView.setText(mPattern);
getPatternRows();
}

@Override
public void onTextChanged(CharSequence cs, int start,
int end, int count) {
int start = mRowEditorView.getTextStart();
getSelectionStart();
mRowEditorView.getText().insert(
start, cs.toString());
}

@Override
public void onTextChange() {
mPattern.notifyDataChanged();
mPattern = mStorage.load(mPattern.getId());
mRowEditorView.setText(mPattern);
getPatternRows();
}

public void savePattern() {
}

```

Listing B.28: RowEditorFragment.java

```

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.View;
15 import android.view.ViewConfiguration;
16 import android.view.ViewGroup;
17 import android.widget.EditText;
18 import android.widget.LinearLayout;
19 import android.widget.TextView;
20 import android.widget.Scroller;
21 import de.muffinworks.knittingapp.R;
22 import de.muffinworks.knittingapp.util.Constants;
23 import de.muffinworks.knittingapp.util.PatternParser;
24 import de.muffinworks.knittingapp.views.LineNumberTextView;
25 import de.muffinworks.knittingapp.views.LineNumberEditText;
26 import de.muffinworks.knittingapp.views.LinedEditorLinearLayout;
27 public class RowEditorLinearLayout extends LinearLayout
28 {
29     private static final String TAG = "RowEditorLinearLayout";
30     private LineNumberEditText lineNumbers;
31     private LineNumberTextView editText;
32     private LinedEditorEditText editText;
33     private boolean mIsBeingDragged = false;
34     private Point mLastScrollTo = new Point();
35     private Scroller mScroller;
36     private Point mLastMotion = new PointF();
37     private VelocityTracker mVelocityTracker;
38     private int mTouchSlop;
39     private int mMinimumVelocity;
40     private int mWidth;
41     private int mMinWidth;
42     public RowEditorLinearLayout(Context context) {
43         super(context);
44         init(context);
45     }
46     public RowEditorLinearLayout(Context context,
47         AttributeSet attrs) {
48         super(context, attrs);
49         init(attrs);
50     }
51     public RowEditorLinearLayout(Context context,
52         AttributeSet attrs, int defStyle) {
53         super(context, attrs, defStyle);
54         init(context);
55     }
56     private void init(Context context) {
57         setOrientation(HORIZONTAL);
58         LayoutInflater inflater = LayoutInflater.from(
59             context);
60         inflater.inflate(R.layout.view_row_editor,
61             true);
62         LineNumbers = (LineNumberTextView) findViewById(
63             R.id.row_editor_line_numbers);
64         editText = (LinedEditorEditText) findViewById(R.
65             id.row_editor_edit_text);
66         editText.addTextChangedListener(new TextWatcher()
67         {
68             @Override
69             public void beforeTextChanged(CharSequence s,
70                 int start, int count, int after) {}
71             @Override
72             public void onTextChanged(CharSequence s,
73                 int start, int count) {}
74             @Override
75             public void afterTextChanged(Editable s) {
76                 scrollToTextChange();
77             }
78         });
79         mScroller = new Scroller(context);
80         setFocusable(true);
81         setDescendantFocusability(FOCUS_AFTER_DESCENDANTS);
82         setWillNotDraw(false);
83         final ViewConfiguration config =
84             ViewConfiguration.get(context);
85         mTouchSlop = config.getScaledTouchSlop();
86         mMinimumVelocity = config.getMinimumFlingVelocity();
87         // ??
88         editText.requestFocus();
89         editText.setSelection(editText.getText().length());
90         @Override
91         protected void onMeasure(int widthMeasureSpec, int
92             heightMeasureSpec) {
93             super.onMeasure(widthMeasureSpec,
94                 heightMeasureSpec);
95             updateEditorLines();
96             mScroller.forceFinished(true);
97             int lineCount = editText.getLineCount();
98             lineNumbers.updateLineNumbers(lineCount);
99             editText.setMinWidth(getWidth());
100            }
101        }
102        public String[] getPattern() {
103            String patternString = editText.getText().toString();
104            return PatternParser.parseRowFormatToPojo(
105                patternString);
106        }
107        public void setPattern(String[] patternRows) {
108            final String pattern = PatternParser.
109            parsePojoToRowFormat(patternRows);
110            // not updating textView when calling setText(

```

```

pattern) - not running on UI thread?
162
163     post(new Runnable() {
164         @Override
165         void run() {
166             editText.setText(pattern);
167         }
168         updateEditorLines();
169     });
170
171     public EditText getEditText() {
172         return editText;
173     }
174     public void disableEditable() {
175         editText.setCursorVisible(false);
176         editText.setFocusableInTouchMode(false);
177         editText.setTextIsSelectable(false);
178         editText.clearFocus();
179     }
180     public void onEnterPressed() {
181         if(editText.getLineCount() > Constants.
182             MAX_ROWS_AND_COLUMNSLIMIT) {
183             Snackbar.make(this, getResources().getString(
184                 R.string.info_max_rows(Constants.
185                 MAX_ROWS_AND_COLUMNSLIMIT), Snackbar.
186                 LENGTH_SHORT).show();
187         } else {
188             editText.dispatchKeyEvent(new KeyEvent(
189                 KeyEvent.ACTION_DOWN, KeyEvent.
190                 KEYCODE_ENTER));
191             updateEditorLines();
192             scrollToTextChange();
193         }
194     }
195     @Override
196     public boolean onTouchEvent(MotionEvent event) {
197         if(!canScroll()) return false;
198         if(mVelocityTracker == null) {
199             mVelocityTracker = VelocityTracker.obtain();
200         }
201         mVelocityTracker.addMovement(event);
202         final int action = event.getAction();
203         final float x = event.getX();
204         final float y = event.getY();
205         switch(action) {
206             case MotionEvent.ACTION_DOWN:
207                 if(!interruptFling)
208                     mScroller.isFinished();
209                 abortAnimation();
210                 mLastMotion.set(x, y);
211
212             break;
213             case MotionEvent.ACTION_MOVE:
214                 deltaX = (int)(mLastMotion.x - x);
215                 deltaY = (int)(mLastMotion.y - y);
216                 mLastMotion.set(x, y);
217                 if(deltaX < 0) {
218                     if(getScrollX() < 0)
219                         deltaX = 0;
220                 }
221                 else if(deltaX > 0) {
222                     final int rightEdge = getWidth() -
223                         getPaddingRight();
224                     final int availableToScroll =
225                         getChildAt(1).getRight() -
226                         getScrollX() - rightEdge;
227                     if(availableToScroll > 0) {
228                         deltaX = Math.min(
229                             availableToScroll, deltaX);
230                     }
231                 }
232                 else if(deltaY < 0) {
233                     if(getScrollY() < 0)
234                         deltaY = 0;
235                 }
236                 else if(deltaY > 0) {
237                     final int bottomEdge = getHeight() -
238                         getPaddingBottom();
239                     final int availableToScroll =
240                         getChildAt(0).getBottom() -
241                         getScrollY() - bottomEdge;
242                     if(availableToScroll > 0) {
243                         deltaY = Math.min(
244                             availableToScroll, deltaY);
245                     }
246                 }
247             break;
248             case MotionEvent.ACTION_UP:
249                 final VelocityTracker velocityTracker =
250                     mVelocityTracker;
251                     velocityTracker.computeCurrentVelocity(
252                         1000);
253                     int initialXVelocity = (int)
254                         velocityTracker.getXVelocity();
255                     int initialYVelocity = (int)
256                         velocityTracker.getYVelocity();
257                     if((Math.abs(initialXVelocity) + Math.
258                         abs(initialYVelocity)) >
259                         mMinimumVelocity) && getChildCount() -
260                         > 0) {
261                         fling(-initialXVelocity, -
262                             initialYVelocity);
263                     }
264     }

```

```

258     mIsBeingDragged = false;
259     break;
260   }
261   // only intercept motion events if we are
262   // dragging
263   return mIsBeingDragged;
264 }
265 @Override
266 protected int computeVerticalScrollRange() {
267   return getChildCount() == 0 ? getHeight() :
268   getChildrenDimensions().bottom;
269 }
270 @Override
271 protected int computeHorizontalScrollRange() {
272   return getChildCount() == 0 ? getWidth() :
273   getChildrenDimensions().right;
274 }
275 @Override
276 protected void measureChild(View child, int
277   parentWidthMeasureSpec, int
278   parentHeightMeasureSpec) {
279   ViewGroup.LayoutParams lp = child.
280   getLayoutParams();
281   child.setLayoutParams(lp);
282   child.measure(childWidthMeasureSpec,
283   childHeightMeasureSpec);
284 }
285 @Override
286 protected void measureChildWithMargins(View child,
287   int widthUsed, int
288   parentWidthMeasureSpec, int
289   parentHeightMeasureSpec) {
290   ViewGroup.LayoutParams lp = child.getLayoutParams();
291   lp.width = widthUsed;
292   child.setLayoutParams(lp);
293   child.measure(childWidthMeasureSpec +
294   childHeightMeasureSpec);
295 }
296 @Override
297 public void computeScroll() {
298   if (mScroller != null) {
299     mScroller.computeScrollOffset();
300   }
301 }
302 @Override
303 public void onInterceptTouchEvent(MotionEvent ev) {
304   final int action = ev.getAction();
305   if ((action == MotionEvent.ACTION_MOVE) &&
306       mIsBeingDragged) {
307     return true;
308   }
309   if (!canScroll()) {
310     mIsBeingDragged = false;
311     return false;
312   }
313   final float x = ev.getX();
314   final float y = ev.getY();
315   switch (action) {
316     case MotionEvent.ACTION_MOVE:
317       final int xDiff = (int) Math.abs(x -
318         mLstMotion.x);
319       final int yDiff = (int) Math.abs(y -
320         mLstMotion.y);
321       if (xDiff > mTouchSlop || yDiff >
322         mTouchSlop) {
323         mIsBeingDragged = true;
324       }
325     break;
326     case MotionEvent.ACTION_DOWN:
327       mLstMotion.x = x;
328       mLstMotion.y = y;
329       mIsBeingDragged = !mScroller.isFinished();
330   }
331 }
332 @Override
333 public void onReleaseDrag() {
334   if (mScroller != null) {
335     mScroller.abortAnimation();
336   }
337 }
338 
```

```

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 = scrollTo((clamp(x, getWidth()) -
306             getPaddingRight()) - getPaddingLeft() -
307             clamp(y, getHeight()) - getPaddingBottom() -
308             getPaddingTop(), childrenDimens.width(),
309             childrenDimens.height());
310     } else {
311         if (oldX != getScrollX() || oldY != getScrollY())
312             onScrollChanged(getScrollX(), getScrollY()
313             (), oldX, oldY);
314     }
315     // Keep on drawing until the animation has
316     // finished.
317     postInvalidate();
318     private Rect getChildrenDimensions() {
319         int childCount = getChildCount();
320         if (childCount > 0) {
321             int width = 0;
322             int height = getChildAt(0).getHeight();
323             for (int i = 0; i < childCount; i++)
324                 View child = getChildAt(i);
325             width += child.getWidth();
326         }
327         return new Rect(0, 0, width, height);
328     }
329     return null;
330 }
331
332 @Override
333 protected void onLayout(boolean changed, int l, int
334 t, int r, int b) {
335     super.onLayout(changed, l, t, r, b);
336     // initial clam
337     scrollTo(getScrollX(), getScrollY());
338 }
339 public void fling(int velocityX, int velocityY) {
340     if (getChildCount() > 0) {
341         int height = getHeight() - getPaddingBottom
342             () - getPaddingTop();
343         int bottom = getChildAt(0).getHeight();
344         int width = getWidth() - getPaddingLeft() -
345             getPaddingRight();
346         int right = getChildAt(1).getRight();
347         mScroller.fling(getScrollX(), getScrollY(),
348             velocityX, velocityY, 0, right - width,
349             0, bottom - height);
350     }
351     invalidate();
352     // we rely on the fact View.scrollBy calls
353     // scrollTo
354     if (getChildCount() > 0) {
355         Rect childrenDimens = getChildrenDimensions
356             ();
357         x = clamp(x, getWidth() - getPaddingRight()
358             () - getPaddingLeft(), childrenDimens.width
359             ());
360         y = clamp(y, getHeight() - getPaddingTop()
361             () - getPaddingBottom(), childrenDimens.
362             width);
363     }
364 }
365 private int clamp(int currentPos, int viewDimens,
366 int childDimens) {
367     if (viewDimens >= childDimens || currentPos < 0)
368         return 0;
369     if (viewDimens + currentPos > childDimens) {
370         childDimens = viewDimens - viewDimens;
371     }
372     return currentPos;
373 }
374 /**
375 * scroll behavior so far: checking if point is in
376 * visible area works and scrolls to point if it
377 * is not visible. edge behavior is faulty:
378 * scrollTo() will clamp if line gets added
379 * at the bottom, will only scroll to second to
380 * last line, since textview height has not been adjusted yet
381 * at that point editText#getCursorPosition()
382 * gives wrong x position if character is added
383 * on last word from "#@string/ lorem long-with-breaks". I believe that is
384 * because a normal editText is made to wrap at the end of it's width. Since I
385 * am suppressing that behavior, and setting
386 * the editText's minimum width new after each
387 * interaction by calling #updateEditorLines}, it
388 * is likely that the implementation gets confused. The
389 * x position that is returned is always
390 * where the added character would be if we wrapped
391 */

```

Listing B.29: RowEditorLinearLayout.java

```

success.exportPattern, 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 (resultCode == Constants.REQUEST_CODE_EDITOR)
    if (requestCode == Activity.RESULT_OK) {
        // user changed pattern and saved ->
        // viewer needs to refresh data
        mPattern = mStorage.load(mPattern.getId()
());
        mRowPattern.setPattern(mPattern.
getPatternRows());
        getActionBarTitle(mPattern.getName());
    } else if (resultCode == Activity.
RESULT_CANCELED) {
        if (data != null) {
            boolean wasPatternDeleted = data.
getBoolean(Boolean.EXTRA_BOOLEAN_EXTRA);
            if (wasPatternDeleted) {
                finish();
            }
        }
    }
}

private void initCounter() {
    mRowText = (TextView) findViewById(R.id.row_
updateRowCounter());
    ImageButton mIncreaseRow = (ImageButton)
findViewById(R.id.button_increase);
    mIncreaseRow.setOnClickListener(new View.
OnClickListener() {
        @Override
        public void onClick(View v) {
            updateRowCounter(mCurrentRow + 1);
        }
    });
}

ImageButton mDecreaseRow = (ImageButton)
findViewById(R.id.button_decrease);
mDecreaseRow.setOnClickListener(new View.
OnClickListener() {
    @Override
    public void onClick(View v) {
        updateRowCounter(mCurrentRow - 1);
    }
});

private void updateRowCounter(int rows) {
    int maxRows = mPattern == null ? Constants.
DEFAULT_ROWS : mPattern.getRows();
    mCurrentRow = Math.min(Math.max(rows, 1),
maxRows);
    mRowText.setText(Integer.toString(mCurrentRow));
    if (mPattern != null) {
        mPattern.setCurrentRow(mCurrentRow);
        mStorage.save(mPattern);
    }
    if (mGridPattern != null) {
        mGridPattern.setCurrentRow(mCurrentRow);
    }
}

private void updateRowCounter() {
    if (mPattern != null) {
        mCurrentRow = mPattern.getCurrentRow();
    }
    updateRowCounter(mCurrentRow);
}

private void initEditors() {
    mPatternContainer = (FrameLayout) findViewById(R.
.id.editor_container);
    mGridPattern = new PatternGridView(this);
    mGridPattern.setCanBeEdited(false);
    mGridPattern.setPattern(mPattern.getPatternRows()
());
    mRowPattern = new RowEditorLinearLayout(this);
    mRowPattern.setEditable(false);
    mPatternContainer.addView(mGridPattern);
    mPatternContainer.addView(mRowPattern);
}

private void switchEditors() {
    if (!mRowFormatActive) {
        mPatternContainer.removeView(mRowPattern);
        mPatternContainer.addView(mRowPattern);
        mRowPattern.setPattern(mPattern.
getPatternRows());
    } else {
        mPatternContainer.removeViewAllViews();
        mPatternContainer.addView(mGridPattern);
        mGridPattern.setPattern(mPattern.
getPatternRows());
    }
}

private void initView() {
    mRowText = (TextView) findViewById(R.id.row_
updateRowCounter());
    ImageButton mIncreaseRow = (ImageButton)
findViewById(R.id.button_increase);
    mIncreaseRow.setOnClickListener(new View.
OnClickListener() {
        @Override
        public void onClick(View v) {
            updateRowCounter(mCurrentRow + 1);
        }
    });
}

```

```

173     mIsRowFormatActive = !mIsRowFormatActive;
174 }
175 }
```

Listing B.30: 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/
4     android:layout_width="match_parent"
5     android:layout_height="match_parent">
6
7 <FrameLayout
8
```

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3   xmlns:android="http://schemas.android.com/apk/res/
4     android:orientation="vertical"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent">
7
8 <ListView
```

```

9   android:id="@+id/glossary_listview"
10  android:layout_width="match_parent"
11  android:layout_height="match_parent"
12 </android.support.design.widget.CoordinatorLayout>
```

Listing B.31: activity_editor.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.design.widget.CoordinatorLayout
3   android:id="@+id/coord"
4   xmlns:android="http://schemas.android.com/apk/res/
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:layout_gravity="bottom"
8     android:layout_margin="0dip/fab-margin"
9     android:fitsSystemWindows="true"
10    tools:context="de.mufinworks.knittingapp.
11    PatternListActivity">
12
13 <ListView
14   android:id="@+id/patterns_list"
15   android:layout_width="match_parent"
16
```

```

17  android:layout_height="match_parent"
18  android:paddingBottom="0dp"
19  android:clipToPadding="false" />
20 <android.support.design.widget.FloatingActionButton
21   android:id="@+id/fab"
22   android:layout_width="wrap_content"
23   android:layout_height="wrap_content"
24   android:layout_gravity="bottom"
25   android:layout_margin="0dip/fab-margin"
26   android:src="@drawable/ic-add-white-24dp" />
27 </android.support.design.widget.CoordinatorLayout>
```

Listing B.32: activity_glossary.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3   xmlns:android="http://schemas.android.com/apk/res/
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:background="@color/colorPrimary"
7     android:layout_gravity="center"
8     android:layout_margin="11dp" />
9
10 <FrameLayout
11   android:id="@+id/editor_container"
12   android:textSize="80sp"
13   android:layout_width="match_parent"
14   android:layout_height="match_parent" />
15 <FrameLayout
16   android:id="@+id/listview_container"
17   android:layout_width="match_parent"
18   android:layout_height="match_parent" />
19 <FrameLayout
20   android:id="@+id/footer_container"
21   android:layout_width="match_parent"
22   android:layout_height="33dp" />
23 <FrameLayout
24   android:id="@+id/footer_bar"
25   android:layout_width="match_parent"
26   android:layout_height="33dp" />
```

Listing B.33: activity_pattern_list.xml

```

24     android:id="@+id/containerControls"
25     android:layout_height="0dp"
26     android:layout_weight="1"/>
27     android:orientation="horizontal"
28   
```

<LinearLayout

```

29     android:id="@+id/containerRows"
30     android:layout_gravity="left_center_vertical"
31     android:layout_width="0dp"
32     android:layout_height="match_parent"
33     android:background="@color/colorPrimary"
34     android:layout_weight="1"
35     android:gravity="center"
36     android:orientation="vertical">
37   
```

<TextView

```

38     android:layout_width="wrap_content"
39     android:layout_height="wrap_content"
40     android:layout_gravity="top_center_horizontal"
41     android:textColor="@color/offblack"
42     android:textSize="20sp"
43     android:text="Reihe" />
44   
```

<TextView

```

45     android:id="@+id/row"
46     android:layout_width="wrap_content"
47     android:layout_height="wrap_content"
48     android:layout_gravity="bottom_center_horizontal"
49     android:textColor="@color/offblack"
50     android:textSize="170sp"
51     android:text="54" />
52   
```

<LinearLayout

```

53   
```

Listing D.34: activity_Viewel.xmll

Listing B 35: attr xm]

```
1 <?xml version="1.0" encoding="utf-8"?>  
2 <resources>  
3 </resources>  
4  
5 </resources>
```

```
<?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">#4E6E71</color>
<color name="highlight-current-row">#95fffc107</color>
<!-- Text color for a button in a pad. -->
<color name="keyboard-button-text-color">#FF99CC</color>
<!-- Ripple color when a button is pressed in a pad -->
<color name="keyboard-button-ripple-color">#1A000000</color>
<color name="primaryText">#212121</color>
<color name="secondaryText">#727272</color>
<color name="divider">#BBB6B6</color>
<color name="ofWhite">#33dede</color>
<color name="black_01">#33aa11</color>
<color name="black_10">#33557575</color>
<color name="black_30">#33292929</color>
<color name="black_60">#66557575</color>
<color name="black_80">#cc557575</color>
<color name="offblack">#22f2f2</color>
<color name="red_400">#FF3350</color>
</resources>
```

Listings B 36: following

Listing B.37: dialog_set_grid_size.xml

卷之三

Listing B.39: dimens-w820.xml

```
1 <resources>
2   <dimen name="fab_margin">16dp</dimen>
3   <dimen name="row_editor_default_text_size">50sp</dimen>
4
5   <dimen name="activity_horizontal_margin">64dp</dimen>
6   <dimen name="activity_min_width">320dp</dimen>
7
8   <dimen name="row_editor_min_width">320dp</dimen>
9
10  <dimen name="row_editor_max_width">480dp</dimen>
11
```

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

```

28 <Gridview
29   android:id="@+id/keyboard_gridview"
30   android:background="@color/colorAccent"
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/_keyboards_button_background"
51   android:onClick="onDeleteToggled" />
52
53 </LinearLayout>
54
55 </LinearLayout>
56
57 </LinearLayout>

```

Listing B.40: fragment_editor_grid.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3   xmlns:android="http://schemas.android.com/apk/res/
4     android"
4     android:orientation="vertical"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:weightSum="3" />
8
9 <de.muffinworks.knittingapp.layouts.
10   RowEditorLineLayout
11   android:id="@+id/row_editor_container"
12   android:layout_weight="2"
13   android:layout_height="0dp" />
14
15 <FrameLayout
16   android:background="@color/colorPrimary"
17   android:layout_width="match_parent"
18   android:layout_height="3dp" />
19
20 <LinearLayout
21   android:layout_width="match_parent"
22   android:layout_height="0dp"
23   android:weightSum="7" />
24   android:layout_weight="1" />
25
26 <Gridview
27   android:id="@+id/keyboard_gridview"
28   android:background="@color/colorAccent"
29   android:scrollbarSize="10dp"
30   android:fadeScrollbars="false"
31   android:horizontalSpacing="20dp"
32   android:layout_width="0dp"
33   android:layout_height="match_parent"
34   android:layout_weight="3" />
35   android:numColumns="3" />
36
37 <FrameLayout
38   android:background="@color/colorAccentDark"
39   android:layout_width="3dp"
40   android:layout_height="match_parent" />
41
42 <include
43   layout="@+id/view_numpad" />

```

Listing B.41: fragment_editor_row.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/
3   android">
4   xmlns:app="http://schemas.android.com/apk/res-auto"
5   xmlns:tools="http://schemas.android.com/tools"
6   tools:context="de.muffinworks.knittingapp.EditorActivity"
7 >
8 <item
9   android:id="@+id/export_pattern"
10  android:orderInCategory="100"
11  android:title="@string/menu_export_pattern"
12  app:showAsAction="never" />
13 <item
14   android:id="@+id/set_size"
15   android:orderInCategory="100"
16   android:title="@string/menu_grid_size"
17   android:showAsAction="never" />
18 <item
19   android:id="@+id/open_glossary"
20   android:orderInCategory="100"
21   android:title="@string/menu_open_glossary"
22   android:showAsAction="never" />
23 <item
24   android:id="@+id/switch_editor"
25   android:orderInCategory="100"
26   android:showAsAction="never" />
27 </menu>

```

Listing B.42: menu_editor.xml

```

11 <?xml version="1.0" encoding="utf-8"?>
12 <menu xmlns:android="http://schemas.android.com/apk/res/
13   android">
14   xmlns:app="http://schemas.android.com/apk/res-auto">
15 <item
16   android:id="@+id/export_all"
17   android:orderInCategory="100"
18   android:title="@string/menu_export_all"
19   app:showAsAction="never" />
20 </menu>

```

Listing B.43: menu_editor-pattern_list.xml

```

17 <item
18   android:id="@+id/reset_row_counter"
19   android:orderInCategory="100"
20   android:title="@string/menu_reset_counter"
21   app:showAsAction="never" />
22 <item
23   android:id="@+id/open_editor"
24   android:orderInCategory="100"
25   android:title="@string/menu_open_editor"
26   android:showAsAction="never" />
27 <item
28   android:id="@+id/scroll_current_row_to_center"
29   android:orderInCategory="100"
30   android:title="@string/menu_jump_to_current_row"
31   android:showAsAction="always" />
32 <item
33   android:id="@+id/open_glossary"
34   android:orderInCategory="100"
35   android:title="@string/menu_open_glossary"
36   android:showAsAction="never" />

```

```

35      <item
36          android:id="@+id/switch_view_style"
37          android:icon="@drawable/icon_swap_horiz_black_24dp"
38          android:orderInCategory="100"
39          android:title="@string/menu_switch_editor" />
40      </menu>
41      <menu :showAsAction="always" />

```

Listing B.44: menu_viewer.xml

```

1 <resources>
2     <string name="app-name">Knitting App</string>
3     <!-- dialog related-->
4     <string name="dialog_ok">OK</string>
5     <string name="dialog_yes">Yes</string>
6     <string name="dialog_no">No</string>
7     <string name="dialog_cancel">Cancel</string>
8     <string name="dialog_grid_size">Change grid size</string>
9     <string name="dialog_title_pattern_delete">Delete pattern %$?</string>
10    <string name="dialog_title_pattern_name">Pattern name</string>
11    <string name="dialog_title_save_changes">Save changes?</string>
12    <!--Activity titles-->
13    <string name="activity-title-glossary">Glossary</string>
14    <!--errors-->
15    <string name="error_over_max_size">Only %$ supported</string>
16    <string name="error_implement_interface">Could not write to disk</string>
17    <string name="error_file_not_found">Could not find file: %$</string>
18    <string name="error_update_metadata">Could not write metadata</string>
19    <string name="error_saver_pattern">Could not write pattern to disk</string>
20    <string name="error_file_not_mounted">Must be at least 1</string>
21    <string name="error_dimension_zero">External storage-not writable</string>
22    <string name="error_external_storage">External storage is not writable</string>
23    <string name="error_external_storage_not_mounted">External storage not available</string>
24    <string name="error_export">Export failed</string>
25    <string name="error_import_no_json">Import failed: can't read file</string>
26    <string name="success_export_pattern">Successfully exported pattern to folder %$ on SD card</string>
27    <string name="success_export_all">Successfully exported to folder %$ on SD card</string>
28    <string name="success_save_pattern">Saving successful</string>
29    <!--success-->
30    <string name="success_export_pattern">Successfully exported pattern to folder %$ on SD card</string>
31    <string name="success_export_all">Successfully exported to folder %$ on SD card</string>
32    <string name="success_save_pattern">Saving successful</string>
33    <string name="success_save_pattern">Saving successful</string>
34
35     <!--info-->
36     <string name="info_import_pattern_already_exists">This pattern already exists. Do you want to overwrite the existing file?</string>
37     <string name="info_storage_permission">Allow access to storage to export and import patterns</string>
38     <string name="info_max_rows">Only %1d rows are supported</string>
39     <!--fragment tags-->
40     <string name="tag_dialog_fragment_grid_size">
41         <string name="false">dialog_fragment_grid_size</string>
42         <string name="tag_dialog_fragment_edit_name">
43             <string name="tag_dialog_fragment_delete_pattern">
44                 <string name="tag_dialog_fragment_set_name">translatable = "false">dialog_fragment_delete_pattern</string>
45             <string name="tag_dialog_fragment_set_name">translatable = "true">dialog_fragment_set_name</string>
46             <!--menu-->
47             <string name="menu_grid_size">Grid size</string>
48             <string name="menu_switch_editor">Switch view</string>
49             <string name="menu_save">Save</string>
50             <string name="menu_edit_name">Change pattern name</string>
51             <string name="menu_delete_pattern">Delete pattern</string>
52             <string name="menu_reset_counter">Reset counter</string>
53             <string name="menu_open_glossary">Open glossary</string>
54             <string name="menu_open_editor">Edit pattern</string>
55             <string name="menu_jump_to_current_row">Current row</string>
56             <string name="menu_export_all">Export all</string>
57             <string name="menu_export_pattern">Export pattern</string>
58             <string name="menu_import_pattern">Import pattern</string>
59             <string name="rows">rows</string>
60             <string name="columns">columns</string>
61             <string name="placeholder_pattern_name">Pattern name</string>
62             <string name="placeholder_pattern_name">Placeholder pattern name</string>
63             <string name="placeholder_line_numbers">translatable = "false">1\2\3\4\5\6</string>
64         </resources>

```

Listing B.45: strings.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string name="app_name">Knitting App</string>
4   <string name="activity-title_glossary">Glossar</string>
5   <string name="columns">Spalten</string>
6   <string name="dialog_cancel">Abbrechen</string>
7   <string name="dialog_no">Nein</string>
8   <string name="dialog_ok">OK</string>
9   <string name="dialog_title_grid_size">Gittergröße ändern
10  </string>
11  <string name="dialog_title_pattern_delete">Strickmuster
12  <string name="dialog_title_pattern_name">
13  <string name="dialog_title_save_changes">Änderungen speichern?</string>
14  <string name="dialog_yes">Ja</string>
15  <string name="error_export">Fehler beim exportieren</string>
16  <string name="error_external_storage_not_mounted">
17  <string name="error_external_storage_not_writable">
    Externer Speicher kann nicht beschrieben werden!</string>
18  <string name="error_file_not_found">Datei: %1$s konnte
    nicht gefunden werden!</string>
19  <string name="error_load_metadata">File reader konnte
    nicht geschlossen werden</string>
20  <string name="error_must_implement_interface"></string>
21  <string name="error_over_max_size">Nur %1$s unterstützt<
    /string>
22  <string name="error_save_pattern">Muster konnte nicht
    gespeichert werden</string>
23  <string name="error_update_metadata">Metadata konnte
    nicht gespeichert werden</string>
24  <string name="info_import_pattern_already_exists">Dieses
    Muster existiert bereits. Muster überschreiben?</string>
25  <string name="info_max_rows">Nur maximal %1$d Reihen
26  unterstützt</string>
27  <string name="info_storage_permission">Zugriff erlauben
    zum Exportieren und Importieren von Mustern</string>
28  <string name="menu_delete_pattern">Muster löschen</string>
29  <string name="menu_edit_name">Musternamen ändern</string>
30  <string name="menu_export_all">Alle exportieren</string>
31  <string name="menu_export_pattern">Muster exportieren</string>
32  <string name="menu_grid_size">Mustergröße</string>
33  <string name="menu_import_pattern">Muster importieren</string>
34  <string name="menu_jump_to_current_row">Aktuelle Reihe</string>
35  <string name="menu_open_editor">Muster bearbeiten</string>
36  <string name="menu_reset_counter">Zähler zurücksetzen</string>
37  <string name="menu_save">Speichern</string>
38  <string name="menu_switch_editor">Ansicht wechseln</string>
39  <string name="placeholder_line_numbers">translatable="false"</string>
40  <string name="placeholder_pattern_name">Mustername</string>
41  <string name="rows">Reihen</string>
42  <string name="success_export_all">Muster erfolgreich
    nach %1$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>

```

```

18  <style name="AppTheme_PopupOverlay" parent="ThemeOverlay
19  .AppCompat.Light" />
20  <item name="RowTextStyle" parent="@style/Widget
21  .AppCompat.EditText">
22  <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
23  <item name="colorAccent">@color/colorAccent</item>
24  <item name="android:background">@android:color/
    transparent</item>
25  <item name="android:cursorVisible">false</item>
26  <item name="android:fontFamily">sans-serif-light</item>
27  <item name="android:includeFontPadding">false</item>
28  </style>
29  <style name="FontButtonStyle" parent="@android:style/
    Widget.Material.Light.Button.Borderless">
30  <item name="android:layoutWidth">wrap-content</item>
31  <item name="android:layoutHeight">wrap-content</item>
32

```

Listing B.46: strings-de.xml

```

33   <item name="android:background">@drawable/
34     keyboard_button_background</item>
35   <item name="android:fontFamily">sans-serif-light</
36     item>
37   <item name="android:gravity">center</item>
38   <item name="android:includeFontPadding">false</item>
39   <item name="android:minWidth">0dp</item>
40   <item name="android:minHeight">0dp</item>
41   <item name="android:textAllCaps">false</item>
42   <item name="android:textColor">@color/
43     keyboard_button_text_color</item>
44   <style name="NumpadLayoutStyle">
45     <item name="android:layout_height">match_parent</item>
46     <item name="android:layout_width">0dp</item>
47     <item name="android:layout_weight">26</item>
48     <item name="android:paddingTop">12dp</item>
49     <item name="android:paddingBottom">2dp</item>
50     <item name="android:paddingLeft">12dp</item>
51     <item name="android:paddingRight">12dp</item>
52     <item name="android:paddingStart">12dp</item>
53     <item name="android:paddingEnd">12dp</item>
54   </style>
55 </resources>

```

Listing B.47: styles.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <resources xmlns:android="http://schemas.android.com/apk/res
3    <style name="FontButtonStyle.KeyEvent">
4      <item name="FontButtonStyle.Key">
5        <item name="android:layout_margin">4dp</item>
6        <item name="android:textSize">32sp</item>
7      </style>

```

Listing B.48: styles-port.xml

```

1  <resources>
2   <style name="AppTheme.NoActionBar">
3     <item name="windowActionBar">false</item>
4     <item name="windowNoTitle">true</item>
5     <item name="android:windowDrawSystemBarBackgrounds"
6       >true</item>
7   </style>

```

Listing B.49: styles-values-v21.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <de.mufinworks.knittingapp.views.KnittingFontButton
3    xmlns:android="http://schemas.android.com/apk/res/
4    android:style="@style/FontButtonStyle.Key"
5    android:padding="30dp"

```

Listing B.50: view_grid_key.xml

```

10   android:layout_width="75dp"
11   android:layout_height="match_parent"
12   android:layout_margin="16dp"
13   android:textSize="32sp"
14   android:text="K"
15   android:gravity="center" />
16   <TextView
17     android:id="@+id/symbol_description"
18     android:layout_width="match_parent"
19     android:layout_height="match_parent"
20     android:id="@+id/symbol"

```

Listing B.51: view_item_glossary.xml

```

21   android:padding="16dp"
22   android:gravity="left|center"
23   android:textSize="26sp"/>
24 </LinearLayout>

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3   xmlns:android="http://schemas.android.com/apk/res/
4     android:orientation="horizontal"
5     android:layout_width="match_parent"
6     android:layout_height="75dp">
7
8   <TextView
9     android:gravity="center-vertical"
10    android:textSize="25sp"
11    android:layout_gravity="start|center_vertical"
12    android:padding="15dp"
13    android:id="@+id/pattern_name"
14    android:text="@string/placeholder_pattern_name"
15    android:layout_weight="1"
16    android:layout_width="0dp"
17    android:layout_height="match_parent" />
18
19   <ImageButton
20     android:tint="@color/colorAccent"

```

Listing B.52: view_item_list_pattern.xml

```

21   android:background="@drawable/
22     keyboard_button_background"
23   android:id="@+id/button_edit"
24   android:src="@drawable/ic_mode_edit_black_24dp"
25   android:layout_gravity="end|center_vertical"
26   android:layout_height="match_parent" />
27
28   <ImageButton
29     android:tint="@color/colorAccent"
30     android:background="@drawable/
31     keyboard_button_background"
32     android:id="@+id/button_delete"
33     android:src="@drawable/ic_delete_black_24dp"
34     android:layout_gravity="end|center_vertical"
35     android:layout_width="50dp"
36     android:layout_height="match_parent" />
37
38 </LinearLayout>

33   android:layout_width="wrap_content" />
34 <de.muffinworks.knittingapp.views.KnittingFontButton
35   style="@style/FontButtonStyle.Key"
36   android:onClick="onNumPadClick"
37   android:text="5"
38   android:layout_width="wrap_content" />
39 <de.muffinworks.knittingapp.views.KnittingFontButton
40   style="@style/FontButtonStyle.Key"
41   android:onClick="onNumPadClick"
42   android:text="6"
43   android:onClick="onNumPadClick"
44   android:layout_width="wrap_content" />
45 <de.muffinworks.knittingapp.views.KnittingFontButton
46   style="@style/FontButtonStyle.Key"
47   android:onClick="onNumPadClick"
48   android:text="7"
49   android:layout_width="wrap_content" />
50 <de.muffinworks.knittingapp.views.KnittingFontButton
51   style="@style/FontButtonStyle.Key"
52   android:onClick="onNumPadClick"
53   android:text="8"
54   android:layout_width="wrap_content" />
55 <de.muffinworks.knittingapp.views.KnittingFontButton
56   style="@style/FontButtonStyle.Key"
57   android:onClick="onNumPadClick"
58   android:text="9"
59   android:layout_width="wrap_content" />
60 <de.muffinworks.knittingapp.views.KnittingFontButton
61   style="@style/FontButtonStyle.Key"
62   android:onClick="onNumPadClick"
63   android:text="9"
64   android:layout_width="wrap_content" />
65 <TextView
66   android:layout_width="wrap_content"

```

```

67      android:layout_height="wrap_content" />
68      <de.muffinworks.knittingapp.views.KnittingFontButton
69          style="@style/FonButtonStyle:key"
70          android:onClick="onNumPadClick"
71          android:text="0"
72          android:layout_width="wrap_content" />
73      </com.android.calculator2.CalculatorPadLayout>
74  
```

Listing B.53: view_numpad.xml

```

1      <?xml version="1.0" encoding="utf-8"?>
2      <LinearLayout
3          xmlns:android="http://schemas.android.com/apk/res/
4              android"
4              android:layout_width="match_parent"
5              android:layout_height="match_parent">
6          <EditText
7              android:id="@+id/input"
8              android:layout_width="wrap_content" />
9      </LinearLayout>
10 
```

Listing B.54: view_pattern_name_input.xml

```

16      android:text="@string/placeholder_line_numbers" />
17      <de.muffinworks.knittingapp.views.LinedEditorEditText
18          android:paddingRight="50dp"
19          style="@style/RowTextStyle"
20          android:id="@+id/row_editor_edit_text"
21          android:gravity="center_vertical|left"
22          android:textSize="10dip"
23          android:textColor="#000000"/>
24          <row_editor_default_text_size>
25          <row_editor_line_numbers>
26          <row_editor_definemenu>
27      </merge>

```

Listing B.55: 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)