



Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

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.

Furthermore, I would like to thank Nadine Kost, Thilo Ilg, and Angela Thomas for their time and participation in the user interviews and testing.

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	15
5 Android Basics	18
5.1 The Operating System Android	18
5.2 Basic Components of an Android App	19
5.2.1 Activity	19

5.2.2 Actionbar	20
5.2.3 Fragment	21
5.2.4 View	23
5.2.5 Storage	23
6 Implementation	25
6.1 Stitch symbols	25
6.2 Pattern and Parsing between Pattern Formats	26
6.3 Persistent Disk Storage	27
6.4 Displaying a Pattern	28
6.4.1 Grid Format	28
6.4.2 Row Format	31
6.5 Keyboard	33
6.6 Viewer with Row Counter	35
6.7 Editor	36
Editor Fragments	36
6.8 Pattern List	37
6.9 Glossary	37
7 User Test	40
8 Evaluation and Discussion	43
9 Outlook	45
Abbreviations	46
List of Figures	47
Listings	50
Bibliography	52
A User Interviews	i
A.1 Question catalogue	ii
A.2 Interview with Thilo Ilg (2016-05-13)	ii
A.3 Interview with Nadine Kost (2016-05-03)	iii
A.4 Interview with Angela Thomas (2016-06-23)	v

CONTENTS

iii

B Source Code

vi

Chapter 1

Introduction

Since the beginning of the 2000s, knitting has encountered a steady rise in popularity (Lewis 2011). This, Lewis 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 generation of knitters 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 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/> (last accessed 08/11/2016))

²<https://www.youtube.com/> (last accessed 08/15/2016))

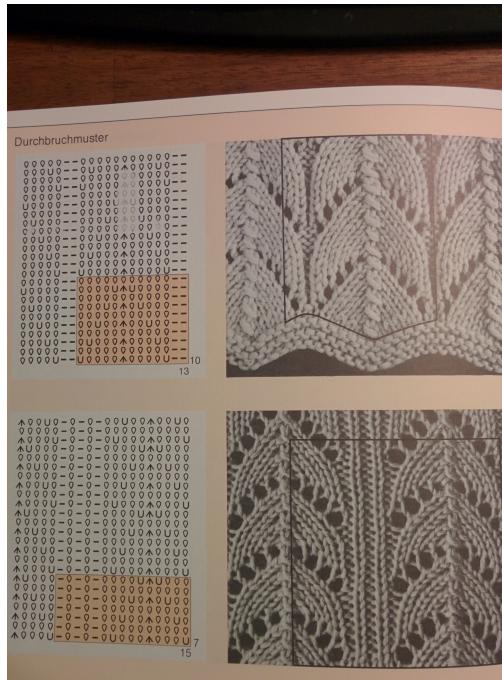


Figure 1.1: Knitting patterns and their corresponding pattern charts from Natter 1983, p142

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 08/11/2016))

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.

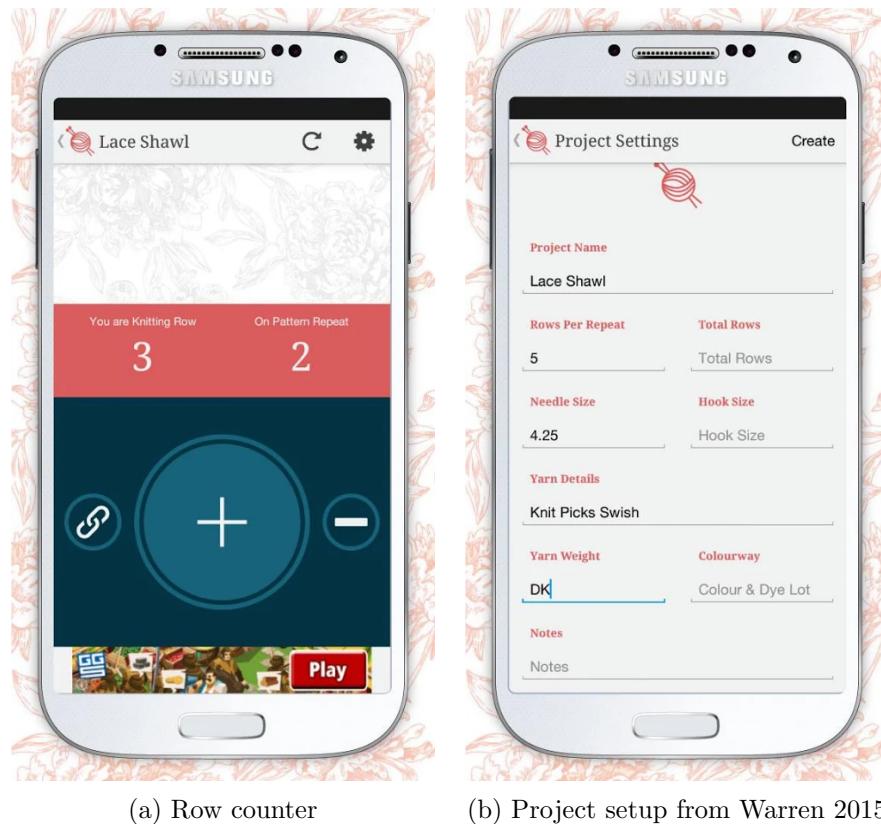
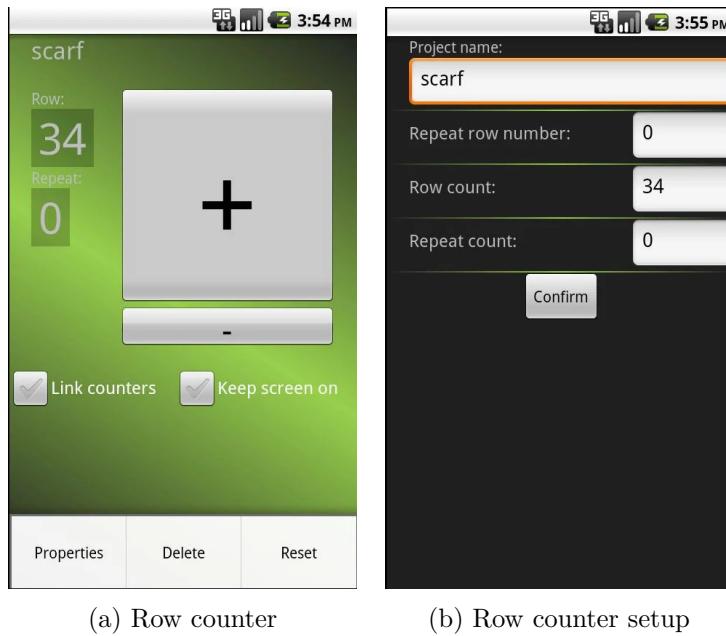


Figure 2.1: Screenshots of the app knit tink

Knitting Counter by mkacki

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

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 08/11/2016))

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.

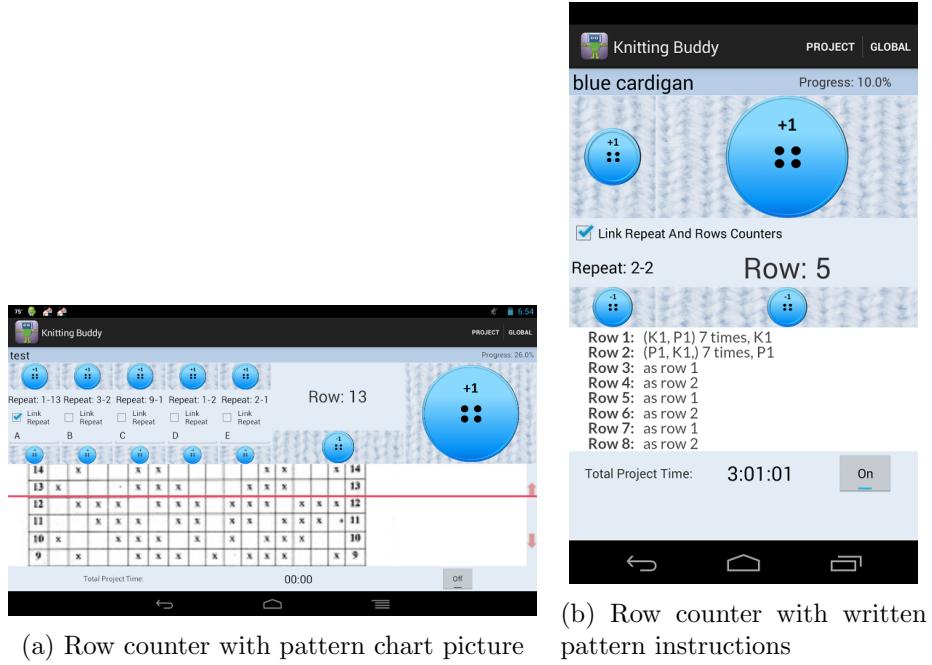
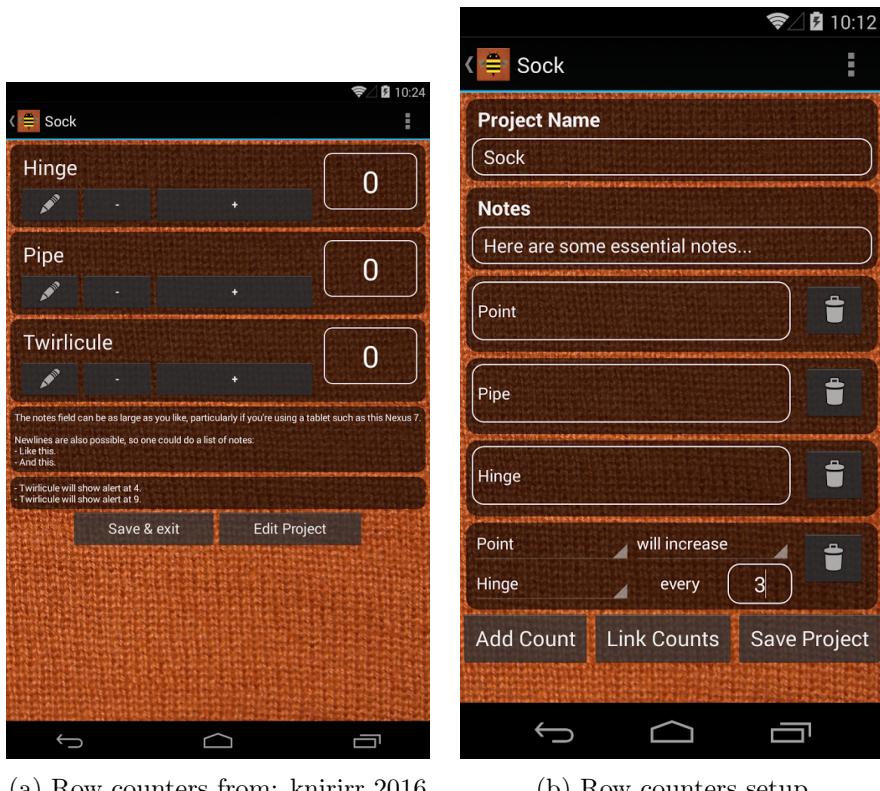


Figure 2.3: Screenshots of the app Knitting and Crochet Buddy

BeeCount knitting Counter by knirirr

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

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 08/11/2016))

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. Exception to this is the app 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 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. Symbols are drawn onto the canvas via OpenGL's drawing methods for images.

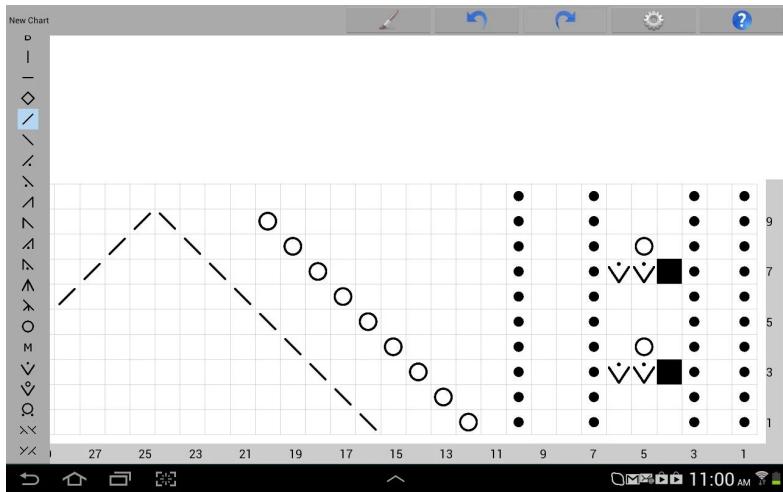


Figure 2.5: Chart editor of Knitting Chart Maker

2.2.2 Other

KnitML by Jonathan Whitall

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

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. The KnitML

project aims to establish an international standard for knitting pattern expressions¹. Whitall aims to do so by using the Knitting Expression Language, 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 KnitML: 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 — the term is used there 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> (last accessed 08/11/2016))

²<http://groovy-lang.org/templating.html> (last accessed 08/11/2016))

Chapter 3

Requirements

3.1 Functional Requirements

The requirements for the app prototype 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 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.

Table 3.1: User stories and functional requirements extracted from the user interviews

#	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

Table 3.1: User stories and functional requirements extracted from the user interviews

#	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 for stitch symbols while inside the app	Glossary of stitches
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

Table 3.1: User stories and functional requirements extracted from the user interviews

#	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 active while the app is open

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 the app is open.

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 the chapter Implementation (Section 6.3). Exported patterns 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 patterns are stored locally on the device and can be manipulated by the user in-app, as CRUD operations are supported. 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, imitating knitting chart patterns printed on paper. 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 predefined 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 input will be inserted and text selection functions for the deletion, copying and pasting of multiple characters 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 located 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. 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 in the app 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. Below the list will be a button to create a new pattern which will open a dialog for entering the new pattern's name.

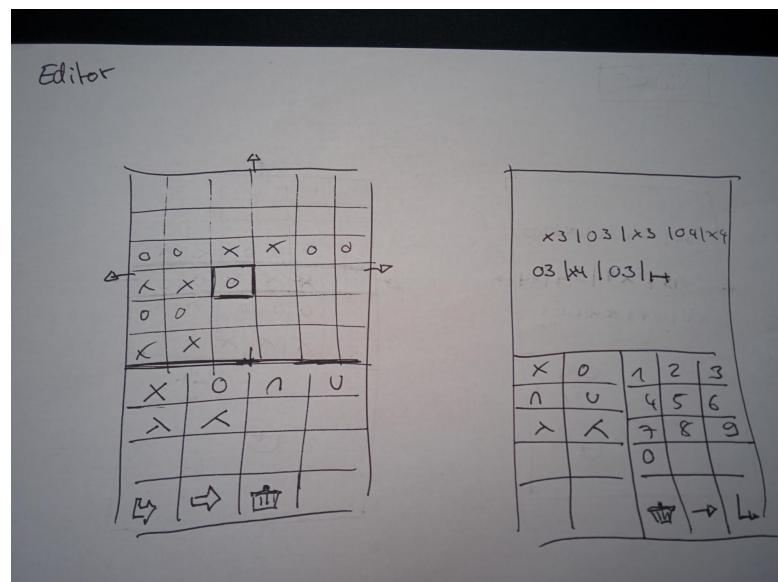


Figure 4.1: Editor screens for grid and row format

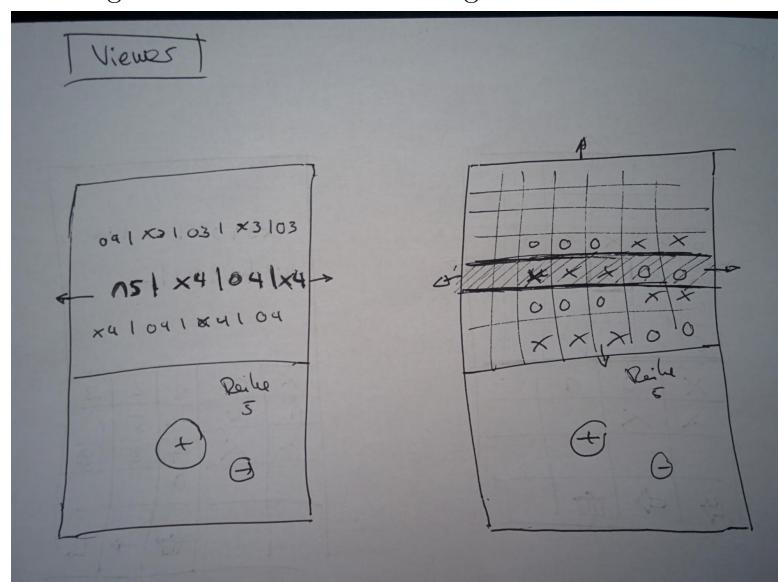


Figure 4.2: Viewer screens for grid and editor format with row counter

After confirming a name, the editor will open the pattern with the default dimensions of 10 columns and 10 rows.

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 digitally 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.

¹<https://developer.android.com/reference/packages.html> (last accessed 08/11/2016)

²<https://developer.android.com/training/index.html> (last accessed 08/11/2016))

³<https://developer.android.com/guide/index.html> (last accessed 08/11/2016))

5.2 Basic Components of an Android App

5.2.1 Activity

The `Activity`⁴ class is needed to display any user interface and as such usually has a single purpose — such as handling a login. An app consists of one or more activities that are in some way connected to each other (Android Developers 2016b). `ViewGroups`⁵ and `Views`⁶ 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 view group inside the activity’s own view hierarchy (Android Developers 2016h). When containing fragments, the activity’s job is to orchestrate the communication between fragments, which is done with callbacks as well as setters 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 and is connected to that activity’s lifecycle.

⁴<https://developer.android.com/reference/android/app/Activity.html> (last accessed 08/11/2016)

⁵<https://developer.android.com/reference/android/view/ViewGroup.html> (last accessed 08/11/2016)

⁶<https://developer.android.com/reference/android/view/View.html> (last accessed 08/11/2016)

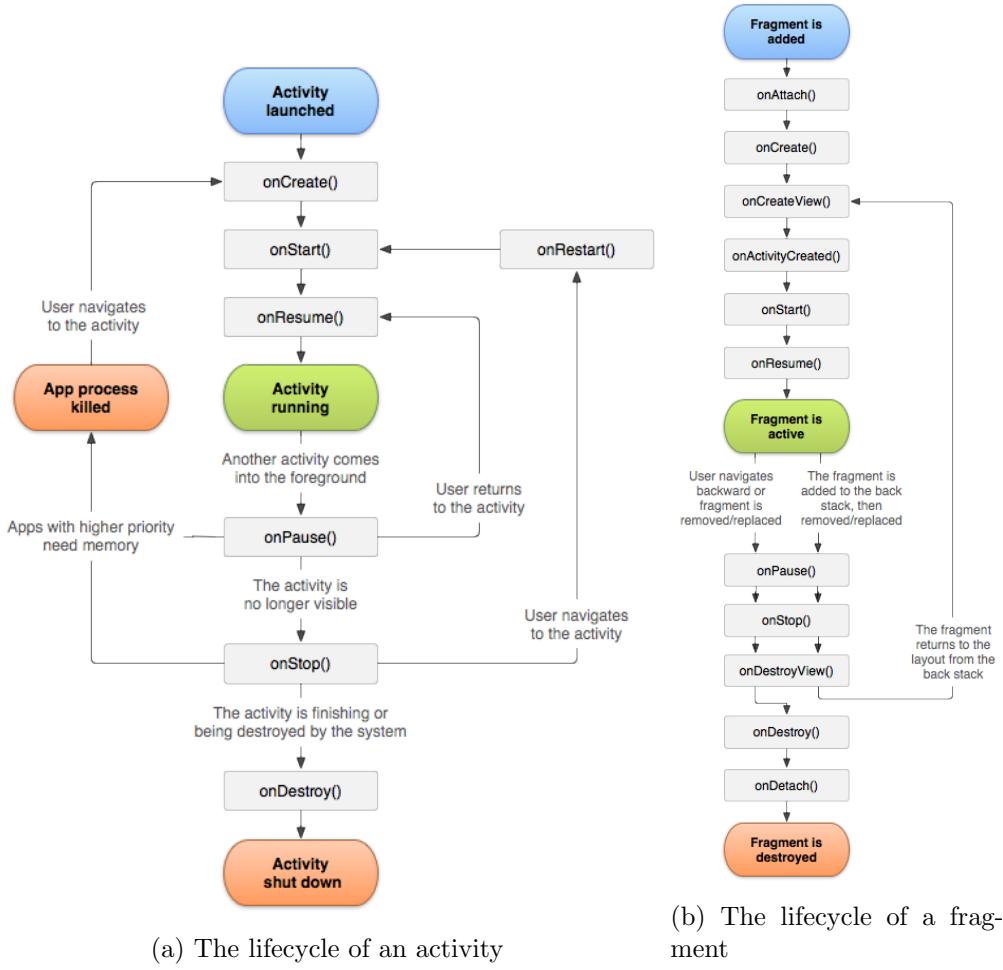


Figure 5.1: The lifecycles of activities and fragments

5.2.2 ActionBar

The **ActionBar** is 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 directly next to buttons that are used frequently, lest the user accidentally

hits it by accident. In the case of an knitting app, such an action could be the deletion of the currently shown pattern.

5.2.3 Fragment

The **Fragment**⁷ class usually implements a specific user interface or behaviour and should, ideally, be modular, so that fragments 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) — which forces the fragment to pause or stop alongside its parent activity's lifecycle. Fragments can also house tree hierarchies of 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 subclasses with predefined behavior, such as the `DialogFragment` class, that opens a fragment as a floating dialog by default (see **Figure 5.3**).

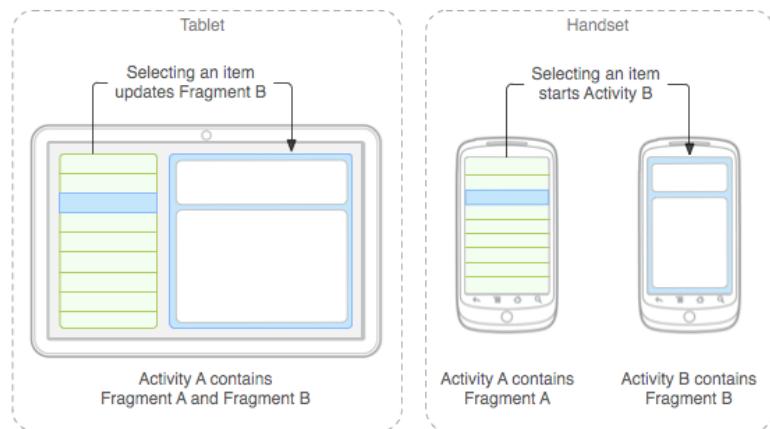


Figure 5.2: Two fragments of one activity and their layout on two different screen sizes

⁷<https://developer.android.com/reference/android/app/Fragment.html> (last accessed 08/11/2016))

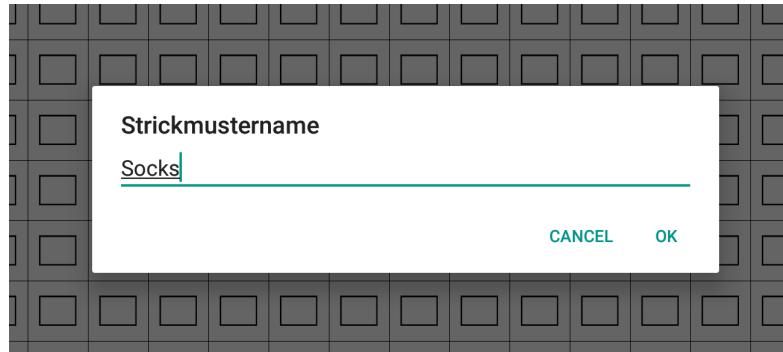


Figure 5.3: A dialog fragment for naming a pattern

Communication between different fragments needs to be handled by their parent activity, which manages the fragments. An activity communicates with a fragment by keeping a reference to that fragment and calling its public methods. The fragment, on the other hand, 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 objects 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 where the point of origin is located at the top left. It is the view's job to handle its own drawing and event handling. For this the view has the predefined methods `onDraw()` and `onTouchEvent()`, respectively. The view class is also the basis for view groups, to which the layouts, view groups specialized in housing other views and view groups, belong to: e.g. `LinearLayout` and `RelativeLayout`.

The views in a window are bundled together as a tree hierarchy with a layout being the top-most root. Views can be added to this hierarchy statically, by specifying them in an 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, the developer 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.

5.2.5 Storage

File storage in Android devices is separated into “internal” and “external” storage. This is due to Android devices often having a built-in, non-removable memory and an external, removable medium in the form of an SD or a microSD 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 placed them in the internal storage. Neither user nor other apps can access them and the files are removed when the app they belong to is uninstalled.

The external storage on the other hand does not require special privileges. Files placed here can be read and written by the user as well as other apps and , if placed

there by an app, they remain even after the app is uninstalled. When working with the external storage it is important to check that the storage is not currently used as Universal Serial Bus (USB) storage by a computer that the device might be connected to.

Apps have by default read and write access to the directory they are installed in inside the internal storage, but to access the external storage it is necessary that the user grants the app specific read and write permissions. These permissions need 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 wants to obtain.

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 that version the app was given all permissions listed in its manifest upon agreeing to install the app from the Play Store. Dangerous permissions cover access to the user's private data, affecting areas where the user stores their data, or data that belong to other apps (Android Developers 2016m). Since the user can revoke an app's permissions at any given time, the developer needs to take extra steps to ensure that the app keeps running even with missing permissions. This can be done by disabling the features that need dangerous permissions.

Chapter 6

Implementation

Google’s IDE Android Studio 2.1.2 is used for the implementation of the Android app prototype. The target version is 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 more detailed 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 leads 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.

Another option for displaying symbols is to create a custom True Type Font (TTF) with glyphs for stitch symbols which is then 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 different 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 to create the custom TTF knitting font used in the prototype. The knitting font contains a selection of 16 stitch symbol glyphs that were defined by the author. The glyphs and their corresponding UTF-8 characters can be 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**(see Appendix B.4) class in the project.

?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z	I	I	J	^	-	-	a	b	c	d	e	f	g	h
i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	}	}
+	M	M	O	A	N	U														

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**

¹<https://fontforge.github.io/en-US/> (last accessed 08/11/2016))

of strings, where each string represents one row in the chart. For this a **Pattern** (see Appendix B.18) Plain Old 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 that is filled with the string representing an empty stitch — an empty stitch is represented by the string “Z” in the prototype. The class **Pattern** is a subclass of **Metadata.java** (see Appendix B.17), 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 the formats. For that the class **PatternParser.java** (see Appendix B.24) 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 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

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

handles the marshalling and unmarshalling of JSON files to Java objects and vice versa. `Gson` supports the usage of Java generics and can map JSON data to POJOs while maintaining inheritance hierarchies. The corresponding code can be found 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 as `Metadata.json`. 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 compared 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 `GridView`³ 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 lists⁴, as well as a `BaseAdapter` that can be subclassed for a custom handling

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

⁴<https://developer.android.com/reference/android/widget/ListAdapter.html> (last accessed 08/11/2016))

and presentation of data. While this is a fitting solution for displaying symbols in a grid, it does not meet the requirements this project sets for the grid format. It is required that the column and row numbers are displayed next to the grid as axes. These axes should scale and scroll together with the grid, but should not be scrolled outside the 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. 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 native two dimensional scroll on any view except its `WebView`⁵ — 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*⁶ and the corresponding training path⁷ 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 predefined, hardcoded values for cell width and margin. The current version of the `PatternGridView` uses a `SimpleOnGestureListener`⁸ 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

⁵<https://developer.android.com/reference/android/webkit/WebView.html> (last accessed 08/11/2016))

⁶<https://developer.android.com/shareables/training/InteractiveChart.zip> (last accessed 08/11/2016)) A digital copy of this project can also be found on the CD attached to this thesis.

⁷<https://developer.android.com/training/gestures/scale.html#drag> (accessed 08/11/2016))

⁸<https://developer.android.com/reference/android/view/GestureDetector.SimpleOnGestureListener.html> (last accessed 08/11/2016))

and speed. This means, that no matter the velocity of the dragging gesture, the view will not keep moving once the user lifts the finger involved in the gesture. A fling will keep moving the view with a speed and duration exponential to the velocity of the fling gesture, where the duration is calculated by introducing a 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`⁹ 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`¹⁰ with is connected to the view's `onTouchEvent()`. The scaling factor is then clamped at predefined maximum and minimum values and saved in a variable. The scale factor is used during the drawing operation to scale the gridlines, axes, and test. When touching

⁹<https://developer.android.com/reference/android/graphics/PointF.html> (last accessed 08/11/2016))

¹⁰<https://developer.android.com/reference/android/view/ScaleGestureDetector.SimpleOnScaleGestureListener.html> (last accessed 08/11/2016))

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 pattern array. Following this the view is invalidated¹¹ and redrawn 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` widget¹² fulfills most of these requirements. The `EditText` widget inherits from the class `TextView` 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 windows 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.

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

¹²[`https://developer.android.com/reference/android/widget/EditText.html`](https://developer.android.com/reference/android/widget/EditText.html) (last accessed 08/11/2016)

Interaction with an `EditText` widget will also show the on-screen keyboard: the widget's `onClick()` and `onLongClick()` listeners trigger the soft input method. To avoid this the custom widget `KeyboardlessEditText`¹³, written by Danial Goodwin, is used in the prototype. It offers the the same functions as a native Android `EditText` widget, but suppresses the showing of the on-screen keyboard.

The `LinedEditor` is used to display the knitting symbols in rof format and 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`¹⁴ 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

¹³<https://github.com/danialgoodwin/android-widget-keyboardless-edittext> (last accessed 08/11/2016)

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

dimensions and line count before the widget has been completely built. This results in a line count return value of zero, no matter how many lines of text have been set in the widget initially. The line numbers in the row editor then display the lowest line number possible: one. Additionally, the parent layout is not able to enable scrolling for offscreen text content, since the child measurements are incorrect. This could be solved by requesting the `EditText` widget's dimensions at a later time when all views 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 the another issue. Upon adding a new line which would lie outside of the visible area, the row editor should scroll the new line into view. Instead only the line above the new line is scrolled into view. This might be caused by measuring the `EditText` widget 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`¹⁵, 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` (see Appendix B.14), a custom class extending Android’s own `Button`¹⁶ 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 project¹⁷. 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.

¹⁵<https://developer.android.com/reference/android/widget/GridView.html> (last accessed 08/11/2016))

¹⁶<https://developer.android.com/reference/android/widget/Button.html> (last accessed 08/11/2016))

¹⁷https://android.googlesource.com/platform/packages/apps/Calculator/+/refs/tags/android-6.0.1_r7/src/com/android/calculator2/CalculatorPadLayout.java (last accessed 08/11/2016))

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`¹⁸ for the pattern content and below that the row counter UI. On its creation the activity instantiates a `PatternGridView` and a `RowEditorLinearLayout` and sets the data of the viewed pattern on both. The view and the layout can then be switched out at runtime inside their container by adding and removing the required view whenever the user decides to switch between grid and 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.

¹⁸<https://developer.android.com/reference/android/widget/FrameLayout.html> (last accessed 08/11/2016))

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

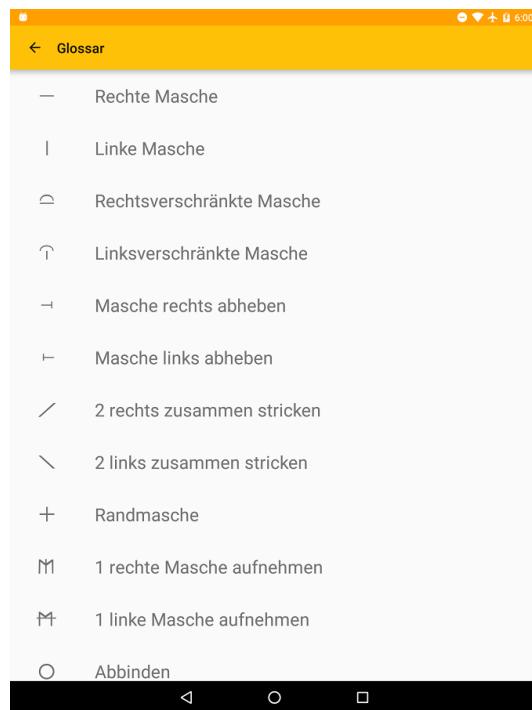
The prototype launches with the `PatternListActivity` (see Appendix B.21) that displays all files currently indexed in the `Metadata` file (see section 6.3). For that Android's `ListView`¹⁹ component is used. For each file the pattern name, a button to edit (pencil icon), and a button to delete (trashcan 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`²⁰ to display the symbols and their descriptions. The symbols and their descriptions are taken from the `Constants` class.

¹⁹<https://developer.android.com/reference/android/widget/ListView.html>

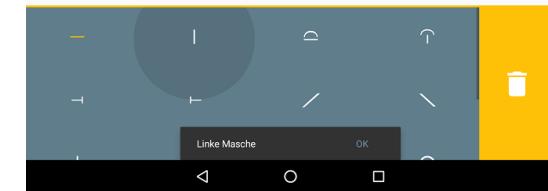
²⁰see footnote 19



(a) The glossary

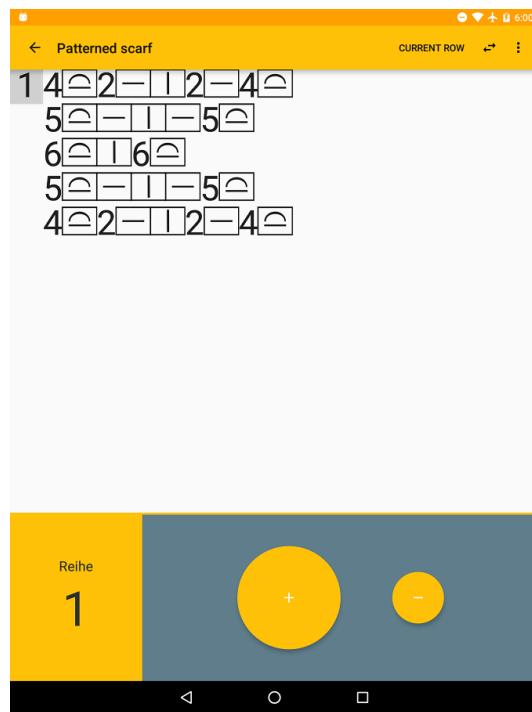
The screenshot shows a grid editor screen titled "Fingerless gloves" with a yellow header bar containing a back arrow and the title. Above the grid is a "GRID SIZE" button. The grid itself is a 10x10 table of symbols, with rows and columns numbered 1 to 10. The symbols represent a pattern for fingerless gloves:

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

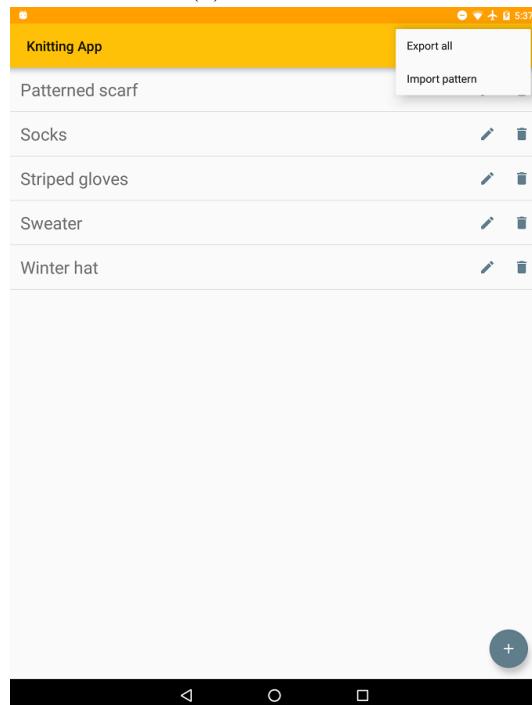


(b) The grid editor while showing a symbol description

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

Except for the row format the prototype fulfills the functional requirements extracted during the user interviews. During testing the users found the prototype to be easy to use after getting to know the row and the different pattern formats. The prototype can therefore be deemed usable.

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 Email or DropBox, would also fit well a knitting 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 08/08/2016)	5
a Row counter at: https://lh6.ggpht.com/-9DvA3pUKqPQwDwi8P_mZX0EhyKz9pE4Dks2QuEKxEGJePvXfY4hUkL00i-zud38c5Y=h900-rw (last accessed 04/04/2016)	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 08/11/2016))	6
a Row counter at: https://lh4.ggpht.com/M05RYCkE7md51ckjB9Bf_CQjz-L6fSS3aWFnQ8UAoURXj04BuHZiWeHtImzAhhpvekE=h900-rw (last accessed 04/04.2016)	6
b Row counter setup at: https://lh3.ggpht.com/AuzeRh7n_r4yLpk9puanH0pBDhcxj6AwC8h5qCaMN3TsRMRu7rML9awuPZTf49M_ejo=h900-rw (last accessed 04/04/2016)	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 04/04/2016)	7

b	Row counter with written pattern instructions at: https://lh3.ggpht.com/EPs72ilPpGCF_fMckHsVb2LeYVx-p6eNjcqg69e0wlsS2h0neneEMEpH29CYH3rEM_c_=h900-rw (last accessed 04/04/2016)	7
2.4	Screenshots of the app BeeCount Knitting Counter at: https://play.google.com/store/apps/details?id=com.knirirr.beeccount (last accessed 08/08/2016)	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 04/04/2016)	8
2.5	Chart editor of Knitting Chart Maker at: https://lh6.ggpht.com/MGKM0ukCD1MWWuboyxmZT-y8P3fTha4SI617u31eK3jFIkLsAllNEA_g6NffaoKRqyg=h900-rw (last accessed 08/11/2016))	9
4.1	Editor screens for grid and row format (own image)	17
4.2	Viewer screens for grid and editor format with row counter (own image)	17
5.1	The lifecycles of activities and fragments	20
a	The lifecycle of an activity at https://developer.android.com/images/activity_lifecycle.png (last accessed 04/04/2016)	20
b	The lifecycle of a fragment at: https://developer.android.com/images/fragment_lifecycle.png (last accessed 08/09/2016)	20
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 08/11/2016))	21
5.3	A dialog fragment for naming a pattern (own image)	22
6.1	The custom knitting font used in the prototype (own image)	26
6.2	Screenshots of the current version of the prototype	38
a	The glossary (own image)	38

b	The grid editor while showing a symbol description (own image)	38
6.3	Screenshots of the current version of the prototype	39
a	The viewer (own image)	39
b	The list of stored patterns (own image)	39

Listings

2.1	Example expression in KnitML	10
2.2	Example expression in KnitML: XML result	10
5.1	Example code for enforcing the implementation of a callback interface . .	22
6.1	Declaring on-screen keyboard hidden in manifest file.	31
6.2	Excerpt from row`editor.xml	32
B.1	AndroidManifest.xml	vii
B.2	BaseActivity.java	vii
B.3	CalculatorPadLayout.java	viii
B.4	Constants.java	ix
B.5	EditorActivity.java	x
B.6	GlossaryActivity.java	xii
B.7	GlossaryAdapter.java	xiii
B.8	GridEditorFragment.java	xiv
B.9	GridSizeDialogFragment.java	xv
B.10	KeyboardAdapterBase.java	xvii
B.11	KeyboardlessEditText2.java	xvii
B.12	KeyboardToggleAdapter.java	xix
B.13	KeyboardTypingAdapter.java	xx
B.14	KnittingFontButton.java	xxi
B.15	LinedEditorEditText.java	xxi
B.16	LineNumberTextView.java	xxii
B.17	Metadata.java	xxii
B.18	Pattern.java	xxii
B.19	PatternDeleteDialogFragment.java	xxiii
B.20	PatternGridView.java	xxiv
B.21	PatternListActivity.java	xxviii

B.22 PatternListAdapter.java	xxx
B.23 PatternNameDialogFragment.java	xxxi
B.24 PatternParser.java	xxxii
B.25 PatternParserTest.java	xxxv
B.26 PatternStorage.java	xxxix
B.27 PatternStorageTest.java	xli
B.28 RowEditorFragment.java	xlii
B.29 RowEditorLinearLayout.java	xliii
B.30 ViewerActivity.java	lviii
B.31 activity_editor.xml	l
B.32 activity_glossary.xml	l
B.33 activity_pattern_list.xml	l
B.34 activity_viewer.xml	l
B.35 attr.xml	li
B.36 colors.xml	li
B.37 dialog_set_grid_size.xml	lii
B.38 dimens.xml	lii
B.39 dimens-w820.xml	lii
B.40 fragment_editor_grid.xml	lii
B.41 fragment_editor_row.xml	liii
B.42 menu_editor.xml	liv
B.43 menu_editor_pattern_list.xml	liv
B.44 menu_viewer.xml	liv
B.45 strings.xml	lv
B.46 strings-de.xml	lvii
B.47 styles.xml	lvii
B.48 styles-port.xml	lvii
B.49 styles-values-v21.xml	lvii
B.50 view_grid_key.xml	lvii
B.51 view_item_glossary.xml	lvii
B.52 view_item_list_pattern.xml	lviii
B.53 view_numpad.xml	lviii
B.54 view_pattern_name_input.xml	lix
B.55 view_row_editor.xml	lix

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 Strickmusterschema arbeiten?*
- Q8. *Könnten Sie sich vorstellen ein Strickmuster von einem mobilen Gerät abzulesen?*
- Q9. *In welchem Format würden Sie dies gerne sehen? (visuell, vom Gerät vorgelesen bekommen)*
- Q10. *Wie würden Sie dem Gerät zu erkennen geben, dass eine Reihe fertig gestrickt wurde? (Sprachbefehl, Knopf drücken)*
- Q11. *(Reihen- und Zellenschreibweise zeigen und nach der Lesbarkeit fragen)*
- Q12. *Haben Sie schon einmal selber Strickmusterschemata 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 (2016-05-13)

- 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 Musterschemas gearbeitet.
- A7. -
- A8. Ja.
- A9. Beides ok, bevorzugt visuell.
- A10. Findet Spracheingabe sehr nützlich, Hände sind beschäftigt beim Stricken.
- A11. Beide Ansichten sind gut, würde sich aber gestört fühlen bei breiten Mustern ständigen Scrollen zu müssen 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 ein Funktion um mehrere Zellen zu markieren und dann mit einem Maschensymbol zu befüllen. Kann sich vorstellen zum auswählen einer Zelle zu klicken. Wenn eine Zelle markiert ist, dann soll nach der Eingabe eines Symbols zur nächsten Zelle gesprungen werden.
- A15. Würde gerne Bilder von dem fertigen Gestrickten sehen und schriftliche Anweisungen bevor er sich mit der Schema befasst. Will für komplexe Muster auf jeden Fall Schemas haben, für simplere Muster eher nicht notwendig. Hätte gerne verschiedenfarbige Symbole für bessere Sichtbarkeit. Wünscht sich Knopf um auf aktuelle Reihe zu springen und einen um die Zoomstufe wieder herzustellen.

A.3 Interview with Nadine Kost (2016-05-03)

- 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 sollte auch ausserhalb der App möglich sein, zum Beispiel wie in Spotifys Android app 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. Bei Tap auf eine Zelle in diesem Raster sollte dann 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 einem Strickmusterschema. Projektmanagement für Strick- projekte. Als Übersetzer von metrischen Einheiten von Nadelgrössen, Gewichten und Längen in imperiale und umgekehrt. Hätte ebenfalls gerne schriftliche An- weisungen in einer App wo man mit Knopfdruck auf nächste Anweisung springen könnte, zB. in Verbindung mit Reihenzähler.

A.4 Interview with Angela Thomas (2016-06-23)

- A1. 49, 72 in einem Jahr war der Rekord. Strickt schon seit vielen Jahren, allerdings keine Muster(zB. Zopf) sondern nur Rechts-Links.
- A2. Größtenteils für Bekannte.
- A3. Stulpen, Dreiecktücher.
- A4. Nein.
- A5. Internet, Strickzeitung, Muster durch Bekannte gelernt.
- A6. Keine Erfahrung mit Musterstricken, hat bisher nur Formschemas für das Häkeln benutzt.
- A7. Für Häkelmuster: mit Stecknadel Reihe markieren.
- A8. Ja.
- A9. Hätte gerne eine Sprachausgabe der momentanen Reihe und würde diese gerne 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 gut 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 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 Musterschema 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 name="de.muffinworks.knittingapp">
5       <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
6         <application
7           android:allowBackup="true"
8             android:icon="@mipmap/ic_launcher"
9               android:label="@string/app_name"
10              android:supportsRtl="true"
11                android:theme="@style/AppTheme">
12                  <activity android:name=".ViewerActivity"/>
13
14                  <activity android:name=".EditorActivity"
15                    android:windowSoftInputMode="
```

```

16
17
18           stateAlwaysHidden"/>
19           <activity android:name=".GlossaryActivity"/>
20             android:name="PatternListActivity"
21               android:label="Knitting App">
22                 <intent-filter>
23                   <action android:name="android.intent.action.MAIN" />
24                     <category android:name="android.intent.category.LAUNCHER" />
25
26           </intent-filter>
27           </activity>
28           </application>
29
30 </manifest>
```

Listing B.1: AndroidManifest.xml

```

37
38   package de.muffinworks.knittingapp;
39
40     import android.Manifest;
41     import android.app.Dialog;
42     import android.content.DalvikVMRuntime;
43     import android.content.Intent;
44     import android.content.pm.ActivityInfo;
45     import android.content.pm.PackageManager;
46     import android.os.Bundle;
47     import android.support.annotation.Nullable;
48     import android.support.v7.app.ActionBar;
49     import android.support.v7.app.AlertDialog;
50     import android.support.v7.app.AppCompatActivity;
51     import android.view.Menu;
52     import android.view.MenuItem;
53     import android.view.View;
54
55     import de.muffinworks.knittingapp.storage.PatternStorage;
56     import de.muffinworks.knittingapp.util.Constants;
57
58     public abstract class BaseActivity extends AppCompatActivity {
59       protected String TAG = this.getClass().getSimpleName();
60
61       @Override
62         protected void onCreate(@Nullable Bundle savedInstanceState) {
63           super.onCreate(savedInstanceState);
64             mStorage = PatternStorage.getInstance();
65             mStorage.init(this);
66
67             mActionBar = getSupportFragmentManager();
68             setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
69
70       }
```

```

38   protected void enableBackInActionBar(boolean enabled)
39     ) {
40       mActionBar.setDisplayHomeAsUpEnabled(enabled);
41       mActionBar.setDisplayShowHomeEnabled(enabled);
42
43     @Override
44       public boolean onOptionsItemSelected(MenuItem item)
45         {
46           if (item.getItemId() == android.R.id.home) {
47             onBackPressed();
48           }
49         }
50
51       return super.onOptionsItemSelected(item);
52
53     protected void setActionBarTitle(String title) {
54       mActionBar.setTitle(title);
55
56     protected boolean isExternalStoragePermissionGranted()
57       {
58         return checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE);
59       }
60
61     @Override
62       protected void onStop() {
63         super.onStop();
64           if (mDialog != null) mDialog.dismiss();
65
66         // https://developer.android.com/training/permissions
67         // requesting.html
68         protected void requestExternalStoragePermission()
69           {
70             if (!isExternalStoragePermissionGranted())
71               if (shouldShowRequestPermissionRationale(
72                 Manifest.permission));
73
74       }
```

```

    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     @Override
57     public boolean shouldDelayPressedState() {
58         return false;
59     }
60
61     @Override
62     protected void onLayout(boolean changed, int left,
63         int top, int right, int bottom) {
64         final int paddingLeft = getPaddingLeft();
65         final int paddingRight = getPaddingRight();
66         final int paddingTop = getPaddingTop();
67         final int paddingBottom = getPaddingBottom();
68
69         final boolean isRTL = ViewCompat.
70             getLayoutDirection(this) ==
71             LAYOUT_DIRECTION_RTL;
72         final int columnWidth =
73             Math.round((float) (right - left) -
74             paddingLeft - paddingRight) /
75             mColumnCount;
76         final int rowHeight =
77             Math.round((float) (bottom - top) -
78             paddingBottom) /
79             mRowCount;
80
81         final MarginLayoutParams lp =
82             getChildLayoutParams();
83         final int childTop = lp.paddingTop + lp.
84             topMargin + rowIndex * rowHeight;
85         final int childBottom = childTop - lp.
86             topMargin - lp.bottomMargin + lp.
87             rowHeight;
88         final int childLeft = paddingLeft + lp.
89             leftMargin + (mColumnCount - 1) -
90             columnIndex * columnWidth;
91         final int childRight = childLeft + lp.
92             columnWidth;
93
94         if (childWidth != childView.getMeasuredWidth)
95             childHeight != childView.getMeasuredWidth
96             || childHeight != childView.getMeasuredHeight()
97             || childWidth != childView.getMeasuredHeight()
98             || childHeight != childView.getMeasuredHeight()
99             || childWidth != childView.getMeasuredWidth
100            || childHeight != childView.getMeasuredHeight()
101            || childWidth != childView.getMeasuredWidth
102            || childHeight != childView.getMeasuredHeight()
103            || childWidth != childView.getMeasuredWidth
104            || childHeight != childView.getMeasuredHeight()
105            || childWidth != childView.getMeasuredWidth
106            || childHeight != childView.getMeasuredHeight()
107            || childWidth != childView.getMeasuredWidth
108            || childHeight != childView.getMeasuredHeight()
109            || childWidth != childView.getMeasuredWidth
110            || childHeight != childView.getMeasuredHeight()
111            || childWidth != childView.getMeasuredWidth
112            || childHeight != childView.getMeasuredHeight()
113            || childWidth != childView.getMeasuredWidth
114            || childHeight != childView.getMeasuredHeight()
115            || childWidth != childView.getMeasuredWidth
116            || childHeight != childView.getMeasuredHeight()
117            || childWidth != childView.getMeasuredWidth
118            || childHeight != childView.getMeasuredHeight()
119            || childWidth != childView.getMeasuredWidth
120            || childHeight != childView.getMeasuredHeight()
121            || childWidth != childView.getMeasuredWidth
122            || childHeight != childView.getMeasuredHeight();
123
124     }

```

ix

```

1 package de.muffinworks.knittingapp.util;
2
3 import android.os.Environment;
4
5 public final class Constants {
6     public static final String EMPTY_SYMBOL = "Z";
7
8     public static String KNITTING_FONT_PATH = "fonts/"
9

```

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.util.Locale;
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             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 }
```

```

42     @Override
43     protected void onCreate(@Nullable Bundle
44         savedInstanceState) {
45         super.onCreate(savedInstanceState);
46         setContentView(R.layout.activity_editor);
47         enableActionBar(true);
48         mPatternId = getIntent().getStringExtra(
49             Constants.EXTRA_PATTERN_ID);
50         if (mPatternId != null) {
51             mPattern = mStorage.load(mPatternId);
52             setActionBarTitle(mPattern.getName());
53         }
54         mRowEditorFragment = RowEditorFragment.
55             getInstance(mPatternId);
56         mGridEditorFragment = GridEditorFragment.
57             getInstance(mPatternId);
58         mFragmentManager = getSupportFragmentManager();
59         FragmentTransaction fm = mFragmentManager.
60             beginTransaction();
61         fm.replace(mFragmentContainer,
62             mGridEditorFragment);
63         fm.commit();
64     }
65     @Override
66     public boolean onCreateOptionsMenu(Menu menu) {
67         getMenuInflater().inflate(R.menu.menu_editor,
68             menu);
69         MenuItemSetGridSize = menu.findItem(R.id.
70             set_size);
71         MenuItemSetGridSize.setVisible(true);
72         return true;
73     }
74     @Override
75     public boolean onOptionsItemSelected(MenuItem item)
76     {
77         int id = item.getItemId();
78         if (id == R.id.set_size) {
79             else if (id == R.id.delete_pattern) {
80                 showDeletePatternDialog();
81                 else if (id == R.id.switch_editor) {
82                     switchEditors();
83                     else if (id == R.id.edit_pattern_name) {
84                         showEditNameDialog();
85                         savePattern();
86                         else if (id == R.id.open_glossary) {
87                             startActivity(new Intent(this,
88                                 GlossaryActivity.class));
89                         else if (id == R.id.export_pattern) {
90                             exportPattern();
91                         return super.onOptionsItemSelected(item);
92                     }
93         private void exportPattern() {
94             try {
95                 mStorage.export(mPatternId);
96                 showAlertDialog(getString(R.string.
97                     success_export_pattern, Constants.
98                     EXPORT_DIR));
99             } catch (IOException e) {
100                 e.printStackTrace();
101             }
102         }
103         @Override
104         public void onBackPressed() {
105             if (wasPatternEdited()) {
106                 AlertDialog saveBeforeExitDialog = new
107                     AlertDialog.Builder(this)
108                     .setTitle(getString(R.string.
109                         dialog_title_pattern_save_changes))
110                     .setPositiveButton(R.string.dialog_yes,
111                         new DialogInterface.OnClickListener()
112                         @Override
113                         public void onClick(DialogInterface dialog,
114                         int which) {
115                             savePattern();
116                             setResult(Activity.RESULT_OK);
117                             finish();
118                         }
119                     .setNegativeButton(R.string.dialog_no,
120                         new DialogInterface.OnClickListener()
121                         @Override
122                         public void onClick(DialogInterface dialog,
123                             int which) {
124                             saveBeforeExitDialog.show();
125                         }
126                     .setResult(!mWasEdited ? Activity.RESULT_OK :
127                         Activity.RESULT_CANCELED : Activity.
128                         RESULT_CANCELED);
129                     }
130                 }
131             }
132             private void switchEditors() {
133                 savePattern();
134                 FragmentTransaction fm = mFragmentManager.
135                 beginTransaction();
136                 if (mRowEditorFragment.isVisible()) {
137                     fm.replace(mFragmentManager,
138                         mGridEditorFragment);
139                 }
140             }
141         }
142     }

```

```

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

Listing B.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 {
```

```

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 KeyboardToggleAdapter;
21 public class GridEditorFragment extends Fragment
22 implements KeyboardToggleAdapter.
23 private static final String BUNDLE_ID = "id";
24 private PatternStorage mStorage;
25 private PatternGridview mPatternGridview;
26 private PatternStorage mStorage;
27 private PatternGridview mPatternGridview;
28 private Gridview mKeyboard;
29 private LinearLayout mDeleteButtonContainer;
30 private KeyboardToggleAdapter mKeyboardAdapter;
31 private boolean mIsDeleteActive = false;
32 private boolean
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
45 @Override
46 public void onCreate(@Nullable Bundle
47 savedInstanceState) {
48     super.onCreate(savedInstanceState);
49     mStorage = PatternStorage.getInstance();
50     if (getArguments() != null) {
51         mStorage.load(getString(BUNDLE_ID));
52     }
53 }
54
55 @Override
56 public View onCreateView(LayoutInflater
57 inflater, ViewGroup container,
58 @Nullable Bundle
59 savedInstanceState) {
60     @Override
61     public void onViewCreated(View view, @Nullable
62     Bundle savedInstanceState) {
63         super.onViewCreated(view, savedInstanceState);
64         mPatternGridView = (PatternGridView) view.
65         findViewById(R.id.gridView);
66         if (mPattern != null) {
67             mPatternGridView.setAdapter(mPattern.
68             getPatternRows());
69         }
70         mDeleteButtonContainer = (LinearLayout) view.
71         findViewById(R.id.
72         grid_delete_button_container);
73         mKeyboard = (GridView) view.findViewById(R.id.
74         keyboard_gridview);
75         mKeyboardAdapter = new KeyboardToggleAdapter(
76             getActivity(),
77             mKeyboard,
78             //set first key active
79             mKeyboardAdapter.getAdapter(),
80             callOnClick());
81         mDeleteButtonContainer.setOnClickListener(new
82         View.OnClickListener() {
83             @Override
84             public void onClick(View v) {
85                 mStorage.setPatternRows(newPatternRows());
86                 mStorage.save(mPattern);
87                 Snackbar.make(v, "Pattern saved!", Snackbar.LENGTH_SHORT).show();
88             }
89         });
90     }
91 }
92
93 public boolean hasPatternChanged() {
94     return !Arrays.deepEquals(mPatternGridView.
95     getPattern(), mPattern.getPatternRows());
96 }
97
98 @Override
99 public void onKeyToggled(String key) {
100     setDeleteActive(false);
101 }
102

```

```

103    public void onDeleteToggled() {
104        setDeleteActive(!mIsDeleteActive);
105    }
106
107    private void setDeleteActive(boolean active) {
108        mIsDeleteActive = active;
109        mKeyboardAdapter.setDeleteActive(mIsDeleteActive);
110        mPatternGridView.setDeleteActive();
111
112        if (mIsDeleteActive)
113            mDeleteButtonContainer.setBackgroundColor(
114                getResources().getColor(R.color.red_400,
115                null));
116    }
117}
118
119    public void setGridSize(int columns, int rows) {
120        mPatternGridView.setGridSize(columns, rows);
121    }
122}



---


114
115    mDeleteButtonContainer.setBackgroundColor(
116        getResources().getColor(R.color.
117        colorPrimary,
118        null));
119
120    mPatternGridView.setGridSize(int columns, int rows);
121
122}



---


114
115    mDeleteButtonContainer.setGridSize(
116        getArguments().getInt(
117            BUNDLE_COLUMNS));
118
119    mPatternGridView.setGridSize(columns, rows);
120
121}



---


114
115    mColumns = getArguments().getInt(
116        BUNDLE_COLUMNS);
117
118    mRows = getArguments().getInt(BUNDLE_ROWS);
119
120}



---


42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

package de.muffinworks.knittingapp.fragments;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.support.v7.app.AlertDialog;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.widget.EditText;
import android.widget.LinearLayout;
import de.muffinworks.knittingapp.R;
import de.muffinworks.knittingapp.util.Constants;
public class GridSizeDialogFragment extends
DialogFragment {
private static final String BUNDLE_COLUMNS =
"columns";
private static final String BUNDLE_ROWS = "rows";
private int mColumns = 0;
private int mRows = 0;
private OnGridSizeInteractionListener mListener;
public GridSizeDialogFragment() {}
public static GridSizeDialogFragment newInstance(int
columns,int rows) {
    GridSizeDialogFragment fragment = new
GridSizeDialogFragment();
    Bundle args = new Bundle();
    args.putInt(BUNDLE_COLUMNS, columns);
    args.putInt(BUNDLE_ROWS, rows);
    fragment.setArguments(args);
    return fragment;
}
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments() != null) {
}
}



---


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

public void onArgumentsChanged() {
    mColumns = getArguments().getInt(
        BUNDLE_COLUMNS);
    mRows = getArguments().getInt(BUNDLE_ROWS);
    mListener.onGridSizeChanged(mColumns, mRows);
}

@Override
public void onGridSizeChanged(int columns, int rows) {
    mListener.onGridSizeChanged(mColumns, mRows);
}

@Override
public void onStart() {
    super.onStart();
    // http://stackoverflow.com/a/15619098/4738174
    final AlertDialog dialog = (AlertDialog)
getDialog();
    if (dialog != null) {
        (dialog.getButton(DialogInterface.BUTTON_NEUTRAL)).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int newColumns = Integer.parseInt(
                    mColumnsEditText.getText());
                int mColumnsChanged = mColumns - newColumns;
                if (newColumns != mColumns || newRows != mRows) {
                    onChartSizeSetResult(
                        Integer.parseInt(
                            mRowsEditText.getText()), toString());
                }
                if (newColumns != mColumns || newRows != mRows) {
                    onChartSizeSetResult(
                        Integer.parseInt(
                            mRowsEditText.getText()), toString());
                }
                Integer.parseInt(
                    mColumnsEditText.getText());
                dismiss();
            }
        });
    }
}

```

```

    string.dialog-title-grid-size))
105   .setView(content)
106     .create();
107   }
108 }
109 private void onChartSizeSetResult(int columns, int
110   rows) {
111   mListener.onSetChartSize(columns, rows);
112   dismiss();
113 }
114 @Override
115 public void onAttach(Context context) {
116   super.onAttach(context);
117   if (context instanceof
118     OnGridSizeInteractionListener) {
119     mListener = (OnGridSizeInteractionListener)
120       context;
121     throw new RuntimeException(context.toString()
122       + getString(R.string.
123         "OnGridSizeInteractionListener"));
124   }
125 }
126 @Override
127 public void onDetach() {
128   super.onDetach();
129 }
130 }
131 mListener = null;
132 public interface OnGridSizeInteractionListener {
133   void onSetChartSize(int columns, int rows);
134 }
135 class DimensionTextWatcher implements TextWatcher {
136   private int oldValue;
137   private EditText editText;
138   public DimensionTextWatcher(EditText editText,
139     private void onTextChanged(CharSequence s,
140       int start, int end, int count) {
141     //Do nothing here;
142     onStart();
143     //Empty onclick needed
144     onOlderVersionsTo
145     instantiateButton();
146   }
147   @Override
148   public void beforeTextChanged(CharSequence s,
149     int start, int count, int after) {
150     @Override
151   public void onTextChanged(CharSequence s, int
152     start, int before, int count) {
153     @Override
154   public void onTextChanged(Editable s) {
155     if (s.length() == 0) {
156       //no input
157       ((AlertDialog) getDialog()).getButton(
158         AlertDialog.BUTTON_POSITIVE).
159       setTitle(getResources().getString(R.
160         title));
161     }
162   }
163 }
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 168
162         error_dimension_zero));
163     editText.setSelection(editText.length());
164     ((AlertDialog)editDialog()).getButton( 169
165         AlertDialog.BUTTON_POSITIVE). 170
166         setEnabled(false);
167     } else if (Integer.parseInt(s.toString()) > 171
168         Constants.MAX_ROWS_AND_COLUMNS_LIMIT) { 172
169     //input > max dimens
170     editText.setError(
171         getString(R.string.error_over_max_size,
172             Constants.
173         .error_over_max_size,
174             Constants.
175         .error_over_max_size));
176 }

```

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

```

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
17 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

```

```

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                         GridEditorKeyListener listener =
191                             (GridEditorKeyListener) v.getTag();
192                         if (listener != null) {
193                             listener.onLongClick(v);
194                         }
195                     }
196                 });
197             convertView.setTag(key);
198         }
199         public KeyboardToggleAdapter(Context context,
200             GridEditorKeyListener listener) {
201             super(context);
202             mListener = listener;
203         }
204         public void setOnLongClickListener(OnLongClickListener
205             listener) {
206             mListener = listener;
207         }
208         public void setOnLongClickListener(OnLongClickListener
209             listener) {
210             mListener = listener;
211         }
212         public void setOnLongClickListener(OnLongClickListener
213             listener) {
214             mListener = listener;
215         }
216         public void setOnLongClickListener(OnLongClickListener
217             listener) {
218             mListener = listener;
219         }
220         public void setOnLongClickListener(OnLongClickListener
221             listener) {
222             mListener = listener;
223         }
224         public void setOnLongClickListener(OnLongClickListener
225             listener) {
226             mListener = listener;
227         }
228         public void setOnLongClickListener(OnLongClickListener
229             listener) {
230             mListener = listener;
231         }
232         public void setOnLongClickListener(OnLongClickListener
233             listener) {
234             mListener = listener;
235         }
236         public void setOnLongClickListener(OnLongClickListener
237             listener) {
238             mListener = listener;
239         }
240         public void setOnLongClickListener(OnLongClickListener
241             listener) {
242             mListener = listener;
243         }
244         public void setOnLongClickListener(OnLongClickListener
245             listener) {
246             mListener = listener;
247         }
248         public void setOnLongClickListener(OnLongClickListener
249             listener) {
250             mListener = listener;
251         }
252         public void setOnLongClickListener(OnLongClickListener
253             listener) {
254             mListener = listener;
255         }
256         public void setOnLongClickListener(OnLongClickListener
257             listener) {
258             mListener = listener;
259         }
260         public void setOnLongClickListener(OnLongClickListener
261             listener) {
262             mListener = listener;
263         }
264         public void setOnLongClickListener(OnLongClickListener
265             listener) {
266             mListener = listener;
267         }
268         public void setOnLongClickListener(OnLongClickListener
269             listener) {
270             mListener = listener;
271         }
272         public void setOnLongClickListener(OnLongClickListener
273             listener) {
274             mListener = listener;
275         }
276         public void setOnLongClickListener(OnLongClickListener
277             listener) {
278             mListener = listener;
279         }
280         public void setOnLongClickListener(OnLongClickListener
281             listener) {
282             mListener = listener;
283         }
284         public void setOnLongClickListener(OnLongClickListener
285             listener) {
286             mListener = listener;
287         }
288         public void setOnLongClickListener(OnLongClickListener
289             listener) {
290             mListener = listener;
291         }
292         public void setOnLongClickListener(OnLongClickListener
293             listener) {
294             mListener = listener;
295         }
296         public void setOnLongClickListener(OnLongClickListener
297             listener) {
298             mListener = listener;
299         }
299     }
300 }
```

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     }
44     @Override
45     public void show() {
46         mListener.onKeyClicked(mCharacters[position]);
47     }
48     @Override
49     public void dismiss() {
50         mListener.onKeyClicked(null);
51     }
52     @Override
53     public void onKeyToggled(int position) {
54         mListener.onKeyToggled(mCharacters[position]);
55     }
56     @Override
57     public void onKeyClicked(View v) {
58     }
59 }

```

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 import android.widget.Button;
7
8 import de.muffinworks.knittingapp.R;
9 import de.muffinworks.knittingapp.util.Constants;
10 public class KnittingFontButton extends Button {
11
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     public KnittingFontButton(Context context,
23         AttributeSet attrs, int defStyleAttr) {
24         super(context, attrs, defStyleAttr);
25     }
26     private void init(Context context) {
27         setTypeface(getKnittingTypeFace(context));
28     }
29     private Typeface getKnittingTypeFace(Context context) {
30         try {
31             Typeface typeface = createFromAsset(context,
32                 Constants.KNITTING_FONT_PATH);
33             public void setActive(boolean mIsActive) {
34                 if (mIsActive) {
35                     setTextColor(getResources().getColor(R.color
36                         .colorPrimary, null));
37                 } else {
38                     setTextColor(getResources().getColor(R.color
39                         .keyboard_button_text_color, null));
40                 }
41             }
42         } catch (Exception e) {
43             e.printStackTrace();
44         }
45     }
46 }

```

Listing B.14: KnittingFontButton.java

```

1 package de.muffinworks.knittingapp.views;
2
3 import android.content.Context;
4 import android.graphics.Point;
5 import android.graphics.Typeface;
6 import android.text.Layout;
7 import android.util.AttributeSet;
8
9 import net.simplyadvanced.widgets.KeyboardlessEditText2;
10 import de.muffinworks.knittingapp.R;
11 import de.muffinworks.knittingapp.util.Constants;
12 import de.muffinworks.knittingapp.util.EditText2;
13
14 public class LinedEditorEditText extends KeyboardlessEditText2 {
15     public LinedEditorEditText(Context context,
16         AttributeSet attrs) {
17         super(context, attrs);
18
19         // set cursor visible and to beginning and
20         // request input focus
21         // needed to update the parents size
22         requestFocus();
23         setSelection(0);
24
25     }
26     private Point getCursorPosition() {
27         try {
28             String url = "https://stackoverflow.com/questions/5044342/how-to
29             -get-cursor-position-x-y-in-edittext-android";
30             Layout layout = getLayout();
31             if (layout != null) {
32                 // check if called before
33                 int pos = getSelectionStart();
34                 int line = layout.getLineForOffset(pos);
35                 int baseline = layout.getLineBaseline(line);
36                 int bl = (int) layout.getPrimaryHorizontal(
37                     pos);
38                 Point test = new Point(bl, baseline);
39             }
40         } catch (Exception e) {
41             e.printStackTrace();
42         }
43     }
44     public void setDimension(R dimen) {
45         rowEditor.defaultTextSize = dimen;
46     }
47     private void setTypeface(Typeface typeface, Context context) {
48         try {
49             getAssets();
50             Constants.KNITTING_FONT_PATH);
51         } catch (Exception e) {
52             e.printStackTrace();
53         }
54     }
55 }

```

Listing B.15: LinedEditorEditText.java

```

1 package de.muffinworks.knittingapp.views;
2
3 import android.content.Context;
4 import android.graphics.Typeface;
5 import android.util.AttributeSet;
6 import android.widget.TextView;
7
8 import de.muffinworks.knittingapp.util.Constants;
9
10 public class LineNumberTextView extends TextView {
11
12     private int lines = 0;
13
14     public LineNumberTextView(Context context) {
15         super(context);
16         setTypeface(Typeface.createFromAsset(context,
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 getColumnns() {
35         return columns;
36     }
37
38     public int getCurrentRow() {
39         return currentRow;
40     }
41
42     public void setCurrentRow(int currentRow) {
43         this.currentRow = currentRow;
44     }
45
46     @Override
47     public boolean equals(Object o) {
48         if (this == o) return true;
49         if (o == null || getClass() != o.getClass()) return false;
50         Pattern pattern = (Pattern) o;
51         return rows.equals(pattern.getRows()) &&
52             columns.equals(pattern.getColumns()) &&
53             currentRow.equals(pattern.getCurrentRow()) &&
54             Arrays.equals(patternRows, pattern.getPatternRows()) &&
55             name.equals(pattern.getName());
56     }
57
58 }
59
60 @Override
61 public int hashCode() {
62     return Objects.hash(patternRows, rows, columns,
63     currentRow, name);
64 }

```

Listing B.18: Pattern.java

```

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

```

```

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 }



---



```

35 private final float ZOOMFACTORMIN = 0.5f;
36 private final float ZOOMFACTORMAX = 2.0f;
37 private final float DEFALTSYMBOLTEXTSIZE = 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 * Based on the tutorial on scroll gesture , dragging ,
53 * and scaling and the example project
54 * Interactivechart
55 * by Google's Android Developers found at
56 * https://developer.android.com/training/gestures/
57 * scroll.html
58 */
59 private float mScaleFactor = 1f;
60 private ScaleGestureDetector mScaleGestureDetector;
61 private GestureDetector mGestureDetector;
62 private PointF mTranslationOffset = new PointF(0, 0);
63 private boolean hasScrolled = false;
64 private String mSelectedSymbol = null;
65 private int mCurrentRow = 0;
66
67 public PatternGridView(Context context) {
68
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.setDither(true);
118    mSymbolPaint.setDither(true);
119    mSymbolPaint.setDither(true);
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 (event .getAction () == MotionEvent .ACTION_UP
220                     && canBeEdited) {
221                 if (mSelectedSymbol != null && !hasScrolled )
222                     float x = event .getX ();
223                     float y = event .getY ();
224                     int row = calculateRowFromValue( y );
225                     int column = calculateColumnFromValue( x );
226
227                     setSymbol( column , row );
228                     postInvalidate();
229                 } else {
230                     hasScrolled = false;
231                 }
232             }
233         }
234         boolean refVal = mScaleGestureDetector .
235             onTouchEvent( event );
236         if (refVal == mGestureDetector .onTouchEvent( event ) ||
237             refVal == super .onTouchEvent( event ) ||
238             refVal == retVal)
239             return refVal;
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                 mScaleFactor ,
252                 MARGIN + CELL_WIDTH / 2 * mScaleFactor ,
253                 MARGIN + row * CELL_WIDTH *
254                 mScaleFactor ,
255                 CELL_WIDTH / 2 * mScaleFactor +
256                 ((int )Math .abs (mSymbolPaint .
257                 getFontMetrics () .top )) / 2
258             );
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         + MARGIN,
325         (rows * CELL_WIDTH *
326           mScaleFactor) + MARGIN,
327         (rows * CELL_WIDTH *
328           mScaleFactor) + MARGIN,
329         mGridPaint);
330     }
331   }
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       width + offset
338       (
339         r * CELL_WIDTH *
340         mScaleFactor +
341         CELL_WIDTH / 2 *
342         mLabelFactor +
343         mLabelTextPaint.
344         measureText(
345         for (int c = 0; c < columns; c++) {
346           int text = c + 1;
347           canvas.drawText(
348             text + "n",
349             c * CELL_WIDTH *
350             mScaleFactor +
351             MARGIN +
352             CELL_WIDTH / 2 *
353             mLabelFactor -
354             mLabelTextPaint.
355             measureText(text
356             +
357             25f - mTranslationOffset.y, // text
358             height + offset
359             mLabelTextPaint);
360   }
361   public void resetZoom() {
362     mScaleFactor = 1.0f;
363     invalidate();
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   }
378 }
```

```

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

```

145     mStorage.save(pattern);
146
147     Intent intent = new Intent(this, EditorActivity.  

148         class);
149     intent.putExtra(Constants.EXTRA_PATTERN_ID,  

150         patternId);
151     intent.putExtra(Constants.EXTRA_PATTERN_ID,  

152         patternId);
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 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343
344 }
345
346 }
347
348 }
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448 }
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
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 }
768 }
769
770 }
771
772 }
773
774 }
775
776 }
777
778 }
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 }
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 }
```

```

108     OnClickListener() {
109         @Override
110         public void onClick(View v) {
111             String patternId = ((Metadata) getItem(
112                 position)).getId();
113             Intent intent = new Intent(mContext,
114                 ViewerActivity.class);
115             intent.putExtra(Constants.
116                 EXTRA_PATTERNID, patternId);
117             mContext.startActivity(intent);
118         }
119     }
120     return convertView;
121 }
122 public void confirmDeleteDialog(final String id,
123     String name) {
124     AlertDialog dialog = new AlertDialog.Builder(
125         mContext)
126         .setTitle(mContext.getString(R.string.delete))
127         .setPositiveButton(R.string.dialog_ok,
128             new DialogInterface.OnClickListener() {
129                 @Override
130                 public void onClick(DialogInterface
131                     dialog, int which) {
132                     PatternStorage storage =
133                         PatternStorage.getInstance();
134                     storage.delete(id);
135                     notifyDataSetChanged();
136     }
137 }
138 }
139 
```

Listing B.22: PatternListAdapter.java

```

26 package de.muffinworks.knittingapp.fragments;
27 import android.app.Dialog;
28 import android.content.Context;
29 import android.content.DialogInterface;
30 import android.os.Bundle;
31 import android.support.annotation.NonNull;
32 import android.support.annotation.Nullable;
33 import android.support.v4.app.DialogFragment;
34 import android.support.v7.app.AlertDialog;
35 import android.text.Editable;
36 import android.text.InputFilter;
37 import android.text.Spanned;
38 import android.text.TextWatcher;
39 import android.widget.EditText;
40 import java.util.regex.Pattern;
41 import de.muffinworks.knittingapp.R;
42 import de.muffinworks.knittingapp.util.Constants;
43 public class PatternNameDialogFragment extends
44     DialogFragment {
45     private static final String BUNDLENAME =
46         "name";
47     
```

```

26 private static final int MAXNAMELENGTH = 45;
27 private OnPatternNameInteractionListener mListener;
28 private String mName = "";
29 public PatternNameDialogFragment() {}
30 
```

```

31 public static PatternNameDialogFragment newInstance(
32     String name) {
33     PatternNameDialogFragment fragment = new
34         PatternNameDialogFragment();
35     Bundle args = new Bundle();
36     args.putString(BUNDLENAME, name);
37     fragment.setArguments(args);
38     return fragment;
39 }
40 
```

```

41 @Override
42 public void onCreate(@Nullable Bundle
43     savedInstanceState) {
44     super.onCreate(savedInstanceState);
45     if (getArguments() != null) {
46         mName = getArguments().getString(BUNDLENAME);
47     }
48 }
```

```

47
48     @NotNull
49     @Override
50     public Dialog onCreateDialog(Bundle savedInstanceState) {
51         final LinearLayout parent = (LinearLayout)
52             getActivity().getLayoutInflater()
53             .inflate(R.layout.view_pattern_name_input,
54                 null);
55         EditText input = (EditText) parent.
56             findViewById(R.id.input);
57         input.setText(mName);
58         input.setSelection(input.length());
59         input.setFilters(new InputFilter[] {
60             new InputFilter.LengthFilter(MAX_NAME_LENGTH) });
61         public void onClick(DialogInterface dialog,
62             int id) {
63             mListener.onSetName(input
64                 .getText().toString());
65         }
66         .setNegativeButton(R.string.dialog_cancel,
67             new DialogInterface.OnClickListener() {
68                 @Override
69                 public void onClick(DialogInterface dialog,
70                     int id) {
71                     dialog.cancel();
72                 }
73                 @Override
74                 public void beforeTextChanged(CharSequence s,
75                     int start, int count, int after) {
76                     @Override
77                     public void onTextChanged(CharSequence s,
78                         int start, int before, int count) {
79                         @Override
80                         public void afterTextChanged(Editable s) {
81                             if (s.toString().isEmpty()) {
82
83
84                     dialog.getButton(DialogInterface.BUTTON_POSITIVE).setEnabled(
85                         false);
86                     } else {
87                         dialog.getButton(DialogInterface.BUTTON_POSITIVE).setEnabled(true
88                         );
89                     });
90                     dialog.setOnShowListener(new DialogInterface.OnShowListener() {
91                         @Override
92                         public void onShow(DialogInterface dialog) {
93                             ((AlertDialog) dialog).getButton(
94                                 AlertDialog.BUTTON_POSITIVE).
95                             setEnabled(false);
96                         });
97                     return dialog;
98                 }
99             }
100            @Override
101            public void onAttach(Context context) {
102                super.onAttach(context);
103                if (context instanceof OnPatternNameInteractionListener) {
104                    mListener = (OnPatternNameInteractionListener)
105                        context;
106                    mListener.onSetNameInteractionListener();
107                }
108            }
109        }
110        @Override
111        public void onDetach() {
112            super.onDetach();
113            mListener = null;
114        }
115    }
116    public interface OnPatternNameInteractionListener {
117        void onSetName(String name);
118    }
119}
120
1
2 package de.muffinworks.knittingapp.util;
3 import java.util.regex.MatchResult;
4 import java.util.ArrayList;
5 import java.util.regex.Matcher;
6 import java.util.regex.Pattern;
7
8 public class PatternParser {
9

```

Listing B.23: PatternNameDialogFragment.java

```

10    private static final String EMPTY = " ";
11    private static final String LINEFEED = "\n";
12    private static final String DEFALTEMPTY.PATTERN.STRING =
13        "\r\n"+Constants.EMPTY_SYMBOL+
14        "\r\n"+Constants.EMPTY_SYMBOL+
15        "\r\n"+Constants.EMPTY_SYMBOL+
16        "\r\n"+Constants.EMPTY_SYMBOL+
17        "\r\n"+Constants.EMPTY_SYMBOL+
18        "\r\n"+Constants.EMPTY_SYMBOL+
19        "\r\n"+Constants.EMPTY_SYMBOL+
20        "\r\n"+Constants.EMPTY_SYMBOL+
21        "\r\n"+Constants.EMPTY_SYMBOL+
22        "\r\n"+Constants.EMPTY_SYMBOL+
23        "\r\n"+Constants.EMPTY_SYMBOL;
24    private static final String REGEX_ALLNUMBER.CHARACTERPAIRS =
25        "((0-9)*)\\b[A-Z]\\b";
26    private static final String REGEX_ALLFORBIDDENCHARS =
27        "[ -+-,!#%&*()";
28    private static final String REGEX_LOOKBEHIND.LINEFEED =
29        "(?<=\\n)";
```

30 static private Pattern pattern = Pattern.compile(

31 REGEX_ALLNUMBER.CHARACTERPAIRS);

32 public static String parseGridToRowFormat(String [] []

33 input) {

34 if (input == null) {

35 String result = "";

36 for (int r = 0; r < input[0].length; r++) {

37 String previousSymbol = input[0][r]; // init

38 String firstSymbol_in_pattern =

39 currentSymbol = input[c][r];

40 int count = 0;

41 for (int c = 0; c < input.length; c++) {

42 currentSymbol = input[c][r];

43 if (currentSymbol.equals(previousSymbol))

44 {

45 count++;

46 } else {

47 // append count and previousSymbol

48 // save new previousSymbol

49 // reset count

50 result += count == 1 ?

51 previousSymbol : count +

52 previousSymbol;

53 previousSymbol = currentSymbol;

54 count = 1;

55 result += count == 1 ? previousSymbol :

56 if (count + previousSymbol == "\n") {

57 result += "\n";

58 }

59 // trim \n from end of string here

60 String test = result.substring(result.length() -

61 1);

62 if ("\n".equals(test)) {

63 result = result.substring(0, result.length() -

64 1);

65 }

66 }

67 public static String [][] parseRowToGridFormat(String

68 input) {

69 if (input == null || input.isEmpty()) return

70 null;

71 // expanded row of 3h -> hh

72 // ArrayList<String> expandedRows = new ArrayList<

73 >();

74 // remove all characters that are not numeric or

75 // used for the symbols font

76 // https://stackoverflow.com/questions/1761051/

77 input = input.replaceAll(

78 REGEX_ALLFORBIDDENCHARS, EMPTY);

79 // split at linefeed \n in to get rows

80 String [] [] compressedRows = input .split(

81 REGEX_LOOKBEHIND.LINEFEED);

82 int columns = 0;

83 for (int r = 0; r < compressedRows.length; r++)

84 for (int c = 0; c < compressedRows[r].length; c++)

85 String row = compressedRows[r].replaceAll(

86 "\d+", EMPTY);

87 row = row.replace("\n", " ");

88 // check if row contained only \n and is now

89 Matcher m = Pattern .matcher (row);

90 if (row.equals(" "))

91 row = Constants.EMPTY_SYMBOL;

92 if (row.equals(" "))

93 row = Constants.EMPTY_SYMBOL;

94 if (row.equals(" "))

95 result += count == 1 ?

96 previousSymbol : count +

97 previousSymbol;

98 previousSymbol = currentSymbol;

99 count = 1;

100 }

101 // find out how many symbols are in the row

```

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}

```

```

194     columnCount += factor;
195     symbolsPerRows[r] += factor;
196     sb.append(factor).append(symbol);
197   } else {
198     columnCount++;
199     symbolsPerRows[r]++;
200     sb.append(symbol);
201   }
202   rows[r] = sb.toString();
203   if (columnCount > columns) columns =
204     columnCount;
205   // Append trailing empty symbols to ensure the
206   // right amount of columns
207   for (int r = 0; r < rows.length; r++) {
208     if (diff == columns - symbolsPerRows[r];
209       rows[r] += diff + Constants.EMPTY_SYMBOL;
210     } else if (diff == 1) {
211       rows[r] += Constants.EMPTY_SYMBOL;
212     }
213   }
214   return rows;
215 }
216 }
217 }
218 public static String[] parseGridFormatCpojo(String
219   gridInput) {
220   String rowFormat = parseGridToRowFormat(
221     gridInput);
222   return parseRowFormatToPojo(rowFormat);
223 }
224 public static String[][] parsePojoToGridFormat(
225   String[][] patternRows) {
226   String[][] patternRows = null || patternRows.length ==
227     Constants.DEFAULT_COLUMNS] = new String[[
228     Constants.DEFAULT_ROWS];
229   for (int c = 0; c < Constants.
230     DEFAULT_COLUMNS; c++) {
231     for (int r = 0; r < Constants.
232       DEFAULT_ROWS; r++) {
233       emptyDefaultPattern[c][r] =
234         Constants.EMPTY_SYMBOL;
235     }
236   }
237 }



---



Listing B.24: PatternParser.java



---



```

27 package de.muffinworks.knittingapp.util;
28 import org.junit.Test;
29 import java.util.ArrayList;
30 import java.util.Arrays;
31 @Test
32 public void convertToGrid_2() {
33 String[] result = PatternParser.
34 parseRowToGridFormat(in);
35 assertTrue(Arrays.deepEquals(result, expected));
36 }
37 @Test
38 public void convertToGrid_2x2() {
39 String[] result = "2h\n2y";
40 String[] expected = {"h", "y"};
41 }
42 @Test
43 public void convertToGrid_2x3() {
44 String[] result = PatternParser.
45 parseRowToGridFormat(in);
46 assertTrue(Arrays.deepEquals(result, expected));
47 }
48 @Test
49 public void convertToGrid_02() {
50 String in = "02h";
51 String[] expected = {"h"};
52 }
53 @Test
54 public void convertToGrid_2x3() {
55 String[] result = "2h\n3y";
56 String[] expected = {"h", "y"};
57 }
58 }

Listing B.25: PatternParserTest.java

```


```

```

52     };
53     String [][] result = PatternParser.
54         parseRowToGridFormat(in);
55     assertTrue(Arrays.deepEquals(result, expected));
56 }
57 @Test
58 public void convertToGrid_2_1() {
59     String in = "2hh";
60     String [][] expected = {
61         {"h", "h"}, {"h", "h"}},
62     };
63     String [][] result = PatternParser.
64         parseRowToGridFormat(in);
65     assertEquals(result, expected);
66 }
67 @Test
68 public void convertToGrid_2x2_empty() {
69     String in = "2.\n2";
70     String [] expected = {
71         {"", ""}, {"", ""}},
72     };
73     String [][] result = PatternParser.
74         parseRowToGridFormat(in);
75     assertEquals(result, expected);
76 }
77 @Test
78 public void convertToGrid_emptyString() {
79     String in = "";
80     String [] expected = null;
81     String [][] result = PatternParser.
82         parseRowToGridFormat(in);
83     assertEquals(result, expected);
84 }
85 }
86 @Test
87 public void convertToGrid_null() {
88     String in = null;
89     String [] expected = null;
90     String [][] result = PatternParser.
91         parseRowToGridFormat(in);
92     assertEquals(result, expected);
93 }
94 }
95 @Test
96 public void convertToGrid_2_1_1x4() {
97     String in = "2hgt\n4d";
98     String [] expected = {
99         {"h", "g", "t"}, {"d", "d", "d"}},
100    };
101   };
102   };
103   };
104   };
105   };
106 }

107 }
108 @Test
109 public void convertToGrid_3x2() {
110     String in = "2h\n2d\n2g";
111     String [][] expected = {
112         {"h", "d"}, {"g", "g"}},
113     };
114     String [][] result = PatternParser.
115         parseRowToGridFormat(in);
116     assertEquals(result, expected));
117 }
118 }
119 @Test
120 public void convertToGrid_3x2_withForbiddenChars() {
121     String in = "2h\n2d\n-#$%&*();/\n>\n?=";
122     String [][] result = PatternParser.
123         parseRowToGridFormat(in);
124     assertEquals(result, expected));
125 }
126 }
127 }
128 }
129 }
130 }
131 @Test
132 public void convertToGrid_2x3_test() {
133     String in = "hdg\nhdg\nh";
134     String [][] expected = {
135         {"d", "g"}, {"d", "g"}, {"h"}},
136     };
137     String [][] result = PatternParser.
138         parseRowToGridFormat(in);
139     assertEquals(result, expected));
140 }
141 }
142 }
143 @Test
144 public void convertToString_3x2() {
145     String in = "h\nh\nh";
146     String [][] expected = {
147         {"h", "h"}, {"h", "h"}, {"h"}},
148     };
149     String expected = "2h\n2d\n2g";
150     String result = PatternParser.
151         parseGridToRowFormat(in);
152     assertEquals(expected.equals(result));
153 }
154 }
155 @Test
156 public void convertToString_3x2WithEmptySpacesInTheEnd() {
157     String in = "h\nh\nh";
158     String [][] expected = {
159         {"h", "h"}, {"h", "h"}, {"h"}},
160     };
161     String expected = "2h\n2d\n2g";
162     String result = PatternParser.
163         parseRowToGridFormat(in);
164     assertEquals(result, expected));
165 }

```

```

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\nn2h"};
241     assertEquals(result == PatternParser.parseGridFormat(in));
242     assertEquals(result == Arrays.deepEquals(result, expected));
243 }
244
245 @Test
246 public void parseRowFormatToPojo() {
247     String[] in = {"", "", "", "", ""};
248     ArrayList<String> result = new ArrayList<>();
249     assertEquals(result.length == expected.size());
250     for (int i = 0; i < result.length; i++) {
251         if (!result[i].equals(expected.get(i))) fail();
252     }
253 }
254
255 @Test
256 public void rowsWithDifferentLengthsToPojo() {
257     String in = "2n\n4g\nn2h";
258     ArrayList<String> expected = new ArrayList<>();
259     expected.add("2n2");
260     expected.add("4g");
261     String[] result = PatternParser.parseRowFormatToPojo(in);
262     assertEquals(result == PatternParser.parseRowFormat(in));
263     assertEquals(result.length == expected.size());
264     for (int i = 0; i < result.length; i++) {
265         if (!result[i].equals(expected.get(i))) fail();
266     }
267 }
268
269 @Test
270 public void gridToPojo() {
271 }

```

```

public void rowToGridWithTrailingNumber() {
    parseRowFormatToPojo(in);
    assertTrue(Arrays.deepEquals(result, expected));
}

String [] [] = {"3r2"};
String [] [] expected = {{}, {"r"}, {"r", "r"}, {"r", "r", "r"}, {"r", "r", "r", "r"}};

String [] [] result = PatternParser.
    parseRowToGridFormat(in);
    assertEquals(Arrays.deepEquals(result, expected));
}

@Test
public void rowToGridEmptyFirstRow() {
    String [] [] in = {"\n3h"};
    String [] [] expected = {{}, {"3", "h"}, {"3h"}};

    String [] [] result = PatternParser.
        parseRowFormatToPojo(in);
        assertEquals(Arrays.deepEquals(result, expected));
}

@Test
public void rowToGridEmptyRow() {
    String [] [] in = "\n";
    String [] [] expected = {{}, {"\n"}, {"\n", "\n"}};

    String [] [] result = PatternParser.
        parseRowFormatToPojo(in);
        assertEquals(Arrays.deepEquals(result, expected));
}

@Test
public void rowToGridEmptyRowInMiddle() {
    String [] [] in = "4f\n\n3h";
    String [] [] expected = {{}, {"4", "f", "3", "h"}, {"4f", "3h"}};

    String [] [] result = PatternParser.
        parseRowFormatToPojo(in);
        assertEquals(Arrays.deepEquals(result, expected));
}

@Test
public void rowToGridEmptyRowAtEnd() {
    String [] [] in = "4f\n\n3h\n";
    String [] [] expected = {{}, {"4", "f", "3", "h"}, {"4f", "3h"}, {"4f", "3h", "\n"}, {"4f", "3h", "\n", "\n"}};

    String [] [] result = PatternParser.
        parseRowFormatToPojo(in);
        assertEquals(Arrays.deepEquals(result, expected));
}

@Test
public void rowToGridMoreThanMaxColumns() {
    String [] [] in = "55h3d";
    String [] [] expected = {{}, {"55", "h", "3", "d"}, {"55h", "3d"}, {"55h3", "d"}, {"55h3d"}, {"55h3d", "\n"}, {"55h3d", "\n", "\n"}};

    String [] [] result = PatternParser.
        parseRowFormatToPojo(in);
        assertEquals(Arrays.deepEquals(result, expected));
}

@Test
public void rowToGridMoreThanMaxColumns() {
    String [] [] in = "5515g";
    String [] [] expected = {{}, {"55", "1", "5", "g"}, {"551", "5g"}, {"5515", "g"}, {"5515g"}, {"5515g", "\n"}, {"5515g", "\n", "\n"}};

    String [] [] result = PatternParser.
        parseRowFormatToPojo(in);
        assertEquals(Arrays.deepEquals(result, expected));
}

```

 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   private static final String TAG = "PatternStorage";
30   private Context mContext;
31   private Gson mGson = new Gson();
32   private HashMap<String, Metadata> mDataTable;
33   private static PatternStorage storage = new
34   PatternStorage();
35   public static PatternStorage getInstance() {
36     if (storage != null) {
37       return storage;
38     } else {
39       storage = new PatternStorage();
40     }
41   }
42   public PatternStorage() {}
43   public String getApplicationDir() {
44     return new PatternStorage();
45   }
46   private PatternStorage() {}
47   public void init(Context context) {
48     this.mContext = context.getApplicationContext();
49     loadMetadata();
50   }
51   private String getApplicationDir() {
52     return Environment.getExternalStorageDir() + "/" + fileName;
53   }
54   private String getFilesDir() {
55     return mContext.getFilesDir() .getPath();
56   }
57   private String getFilePathInApplicationDir( String
58   fileName) {
59     return getApplicationDir() + "/" + fileName;
60   }
61   private boolean isExternalStorageWritable() {
62     return Environment.MEDIA_MOUNTED.equals(
63       Environment.getExternalStorageState()) &&
64       Environment.getExternalStorageDirectory()
65       () .canWrite() ;
66   }
67   public void exportAll() throws IOException {
68     for (String id : mDataTable.keySet()) {
69       export(id);
70     }
71   }
72   public File export(String id) throws IOException {
73     if (!isExternalStorageWritable())
74       throw new IOException(mContext.getString(R.
75         string.error_external_storage_not_mounted));
76     File patternFile = new File(
77       getFilePathInApplicationDir(id + ".json"));
78     File file = new File(Environment.
79       getExternalStorageDirectory());
80     file.mkdirs();
81     file = new File(file , id + ".json");
82     copyFile(patternFile , file);
83   }
84   public void importPattern(String path) throws
85   JsonSyntaxException {
86     save(loadFromFile(path));
87   }
88   /**
89    * From https://stackoverflow.com/questions/9292954/
90    * how-to-make-a-copy-of-a-file-in-android
91    */
92   private void copyFile( File src , File dst ) throws
93   IOException {
94     FileOutputStream outStream = new
95     FileInputStream(inStream =
96       FileInputStream(src));
97     FileChannel inChannel = inStream.getChannel();

```

```

96     FileChannel outChannel = outStream.getChannel() ;
97     inChannel.transferTo(0, inChannel.size(), outChannel);
98     inStream.close();
99     outStream.close();
100    }
101
102    private void loadMetadata() {
103        mMetaDataTable = new HashMap<>();
104
105        try {
106            File file = new File(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     public void delete(Metadata pattern) {
193         getFileFromApplicationDir(pattern, getFilename())
194             .delete();
195         mMetadataTable.remove(pattern.getId());
196         updateMetadata();
197     }
198     public void delete(String id) {
199         delete(mMetadataTable.get(id));
200     }
201 }



---


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



---


23     pattern.setName("scarf");
24     pattern.setCurrentRow(2);
25     pattern.setPatternRows(new String[] {
26         "3e",
27         "2er",
28         "ttt",
29         "3f",
30         "3t",
31     });
32     assertEquals(3, pattern.getColumns());
33     assertEquals(5, pattern.getRows());
34     storage.save(pattern);
35     File test = new File(context.getFilesDir());
36     test.getPath().toString() + "/newPatternName();
37     assertTrue(test.exists());
38     Pattern p2 = storage.load(pattern.getId());
39     assertEquals(pattern.equals(p2), true);
40     assertEquals(storage.listMetadataEntries().length
41         == 0);
42     for(int i = 1; i <= 5; i++) {
43         Pattern pattern = new Pattern();
44         storage.save(pattern);
45         assertTrue(storage.listMetadataEntries().length
46             > 0);
47         String id = storage2.listMetadataEntries()[0];
48         getId();
49         storage.delete(id);
50     }
51 }



---


53     import android.content.Context;
54     import android.os.Environment;
55     import android.support.test.InstrumentationRegistry;
56     import android.support.test.runner.AndroidJUnit4;
57     import org.junit.After;
58     import org.junit.Before;
59     import org.junit.Test;
60     import org.junit.runner.RunWith;
61     import java.io.File;
62     import de.muffinworks.knittingapp.storage.models.
63     Metadata;
64     import de.muffinworks.knittingapp.storage.models.Pattern;
65     import static org.junit.Assert.assertEquals;
66     import static org.junit.Assert.assertNull;
67     import static org.junit.Assert.assertTrue;
68     import static org.junit.Assert.fail;
69     import static org.junit.Assume.assumeTrue;
70     @RunWith(AndroidJUnit4.class)
71     public class PatternStorageTest {
72     }



---


73     private Context context = InstrumentationRegistry.
74     getInstrumentation();
75     private PatternStorage storage;
76     private ApplicationContext context;
77     private Pattern pattern;
78     private PatternStorageTest() throws
79     IOException;
80     before {
81         before {
82             pattern = new Pattern();
83             storage = PatternStorage.getInstance();
84             storage.init(context);
85             storage.clearAll();
86         }
87     }
88     @Test
89     public void saveAndLoadPatternTest() throws
90     IOException {
91         assertNull(storage.load("fieldThatDoesNotExist"));
92         Pattern pattern = new Pattern();
93     }



---


43     pattern.setName("scarf");
44     pattern.setCurrentRow(2);
45     pattern.setPatternRows(new String[] {
46         "3e",
47         "2er",
48         "ttt",
49         "3f",
50         "3t",
51     });
52     assertEquals(3, pattern.getColumns());
53     assertEquals(5, pattern.getRows());
54     storage.save(pattern);
55     File test = new File(context.getFilesDir());
56     test.getPath().toString() + "/newPatternName();
57     assertTrue(test.exists());
58     Pattern p2 = storage.load(pattern.getId());
59     assertEquals(pattern.equals(p2), true);
60     assertEquals(storage.listMetadataEntries().length
61         == 0);
62     for(int i = 1; i <= 5; i++) {
63         Pattern pattern = new Pattern();
64         storage.save(pattern);
65         assertTrue(storage.listMetadataEntries().length
66             > 0);
67         assertEquals(storage.listEntriesTest().length
68             == 0);
69         assertEquals(storage.listMetadataEntries().length
70             > 0);
71     }
72 }



---


73     public void deleteTest() throws IOException {
74         saveAndLoadPatternTest();
75         PatternStorage storage2 = PatternStorage.
76             getInstance();
77         assertTrue(storage2.listMetadataEntries().length
78             > 0);
79         String id = storage2.listMetadataEntries()[0];
80         getId();
81         storage.delete(id);
82     }

```

```

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     }

```

```

40      super.onCreate(savedInstanceState);
41      mStorage = PatternStorage.getInstance();
42      mStorage.initState();
43      if (getArguments() != null) {
44          mPattern = mStorage.load getArguments();
45          getString("id"));
46      }
47      @Nullable
48      @Override
49      public View onCreateView(LayoutInflater inflater,
50          @Nullable ViewGroup container, @Nullable Bundle
51          savedInstanceState) {
52          return inflater.inflate(R.layout.
53              fragment_editor_row, container, false);
54      }
55      @Override
56      public void onViewCreated(View view, @Nullable
57          Bundle savedInstanceState) {
58          super.onViewCreated(view, savedInstanceState);
59          mRowEditorView = (RowEditorLinearLayout) view;
60          findViewById(R.id.row_editor_container);
61          findViewById(R.id.keyboard_gridview);
62          mKeyboard.setAdapter(new KeyboardTypingAdapter(
63              getActivity(), this));
64          view.findViewById(R.id.op_enter).
65          setOnClickListener(new View.OnClickListener {
66              @Override
67              public void onClick(View v) {
68                  onEnter();
69              }
70          });
71          view.findViewById(R.id.op_delete).
72          setOnClickListener(new View.OnClickListener {
73              @Override
74              public void onClick(View v) {
75                  onDelete();
76              }
77          });
78      }
79      public void savePattern() {
80          mPattern.setPatternRows(mRowEditorView.
81              getPattern());
82          mStorage.save(mPattern);
83          Snackbar.make(itemView(), getString(R.string.
84              success_save_pattern), Snackbar.LENGTH_SHORT
85              ).show();
86      }
87      public boolean hasPatternChanged() {
88          return Arrays.deepEquals(mRowEditorView.
89              getPattern(), mPattern.getPatternRows());
90      }
91      /**
92       * emulates the backspace key on the system's soft
93       * keyboard
94       */
95      public void onDelete() {
96          /**
97           * emulates the enter key on the system's soft
98           * keyboard
99       */
100     public void onEnter() {
101     }
102     public void onNumPadClick(String number) {
103         int start = mRowEditorView.getEditText().
104             getSelectionStart();
105         mRowEditorView.getEditText().getText().insert(
106             start, number);
107     }
108     public void notifyDataChanged() {
109         mPattern = mStorage.load(mPattern.getId());
110         mRowEditorView.setPattern(mPattern.
111             getPatternRows());
112     }
113     @Override
114     public void onKeyClicked(String key) {
115         int start = mRowEditorView.getEditText().
116             getSelectionStart();
117         mRowEditorView.getEditText().getText().insert(
118             start, key.toUpperCase());
119     }
120 }

```

Listing B.28: RowEditorFragment.java

```

1 package de.muffinworks.knittingapp.layouts;
2 import android.graphics.PointF;
3 import android.support.design.widget.Snackbar;
4 import android.text.Editable;
5 import android.graphics.Point;
6 import android.content.Context;
7 import android.support.v4.app.Fragment;
8 
```

```

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 lineNumbersView;
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 mMaximumVelocity;
41     public RowEditorLinearLayout(Context context) {
42         super(context);
43         super(context);
44         init(context);
45     }
46     public RowEditorLinearLayout(Context context,
47         AttributeSet attrs) {
48         super(context, attrs);
49         super(attrs);
50         init(attrs);
51     }
52     public RowEditorLinearLayout(Context context,
53         AttributeSet attrs, int defStyle) {
54         super(context, attrs, defStyle);
55         init(attrs);
56     }
57     private void init(Context context) {
58         setOrientation(HORIZONTAL);
59         LayoutInflater inflater = LayoutInflater.from(
60             context);
61         inflater.inflate(R.layout.view_row_editor,
62             true);
63         LineNumbers = (LineNumberTextView) findViewById(
R.id.row_editor_line_numbers);
64         editText = (LinedEditorEditText) findViewById(R.
id.row_editor_edit_text);
65         editText.addTextChangedListener(new TextWatcher()
{
66             @Override
67             public void beforeTextChanged(CharSequence s,
68                 int start, int count, int after) {}
69             @Override
70             public void onTextChanged(CharSequence s,
71                 int start, int count) {}
72             @Override
73             public void afterTextChanged(Editable s) {
74                 scrollToTextChange();
75             }
76         });
77         mScroller = new Scroller(context);
78         setFocusable(true);
79         setDescendantFocusability(FOCUS_AFTER_DESCENDANTS);
80         setWillNotDraw(false);
81         final ViewConfiguration config =
82             ViewConfiguration.get(context);
83         mTouchSlop = config.getScaledTouchSlop();
84         mMinimumVelocity = config.getScaledMinimumFlingVelocity();
85         // ??
86         editText.requestFocus();
87         editText.setSelection(editText.getText().length());
88         @Override
89         protected void onMeasure(int widthMeasureSpec, int
heightMeasureSpec) {
90             super.onMeasure(widthMeasureSpec, heightMeasureSpec);
91             heightMeasureSpec = heightMeasureSpec - heightMeasureSpec;
92             updateEditorLines();
93         }
94         public void updateEditorLines() {
95             mScroller.forceFinished(true);
96             int lineCount = editText.getLineCount();
97             lineNumbers.updateLineNumbers(lineCount);
98             editText.setMinWidth(getWidth());
99             editText.setWidth();
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     });
169     updateEditorLines();
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_COLUMNS_LIMIT) {
183             Snackbar.make(this, getResources().getString(
184                 R.string.info_max_rows(Constants.
185                 MAX_ROWS_AND_COLUMNS_LIMIT), 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) +
258                         mMinumVelocity) >
259                         abs(initialYVelocity) && getChildCount() -
260                         > 0) {
261                         fling(-initialXVelocity, -
262                             initialYVelocity);
263                     }
264     }

```

```

206     if (mVelocityTracker != null) {
207         mVelocityTracker.recycle();
208         break;
209     }
210 }
211
212     } return true;
213
214     private boolean canScroll() {
215         int childCount = getChildCount();
216         if (childCount > 0) {
217             int childrenHeight = 0;
218             int childrenWidth = 0;
219             for (int i = 0; i < childCount; i++) {
220                 View child = getChildAt(i);
221                 childrenHeight += child.getHeight();
222                 childrenWidth += child.getWidth();
223             }
224             return (getHeight() < childrenHeight +
225                     getPaddingTop() + getPaddingBottom() +
226                     ((getWidth() < childrenWidth ? +
227                         getPaddingRight() + getPaddingLeft())));
228         }
229         return false;
230     }
231
232     @Override
233     public boolean onInterceptTouchEvent(MotionEvent ev)
234     {
235         final int action = ev.getAction();
236         if ((action == MotionEvent.ACTION_MOVE) &&
237             (mIsBeingDragged)) {
238             return true;
239         }
240         if (!canScroll())
241             mIsBeingDragged = false;
242         final float x = ev.getX();
243         final float y = ev.getY();
244         switch (action) {
245             case MotionEvent.ACTION_MOVE:
246                 final int xDiff = (int) Math.abs(x -
247                     mLsMotion.x);
248                 final int yDiff = (int) Math.abs(y -
249                     mLsMotion.y);
250                 if ((xDiff > mTouchStop) || yDiff >
251                     mTouchStop) {
252                     mIsBeingDragged = true;
253                 }
254             break;
255             case MotionEvent.ACTION_DOWN:
256                 mLsMotion.x = x;
257                 mLsMotion.y = y;
258                 mIsBeingDragged = !mScroller.isFinished();
259             break;
260         }
261     }
262
263     protected int computeVerticalScrollRange() {
264         return getHeight();
265     }
266     @Override
267     protected int computeHorizontalScrollRange() {
268         return getWidth();
269     }
270     @Override
271     protected int computeChildCount() {
272         return getChildCount();
273     }
274     @Override
275     protected void measureChild(View child, int
276         parentWidthMeasureSpec,
277         parentHeightMeasureSpec,
278         ViewGroup.LayoutParams lp) {
279         lp.heightMeasureSpec = getChildMeasureSpec(
280             parentWidthMeasureSpec,
281             childWidthMeasureSpec,
282             childHeightMeasureSpec);
283         lp.widthMeasureSpec = getChildMeasureSpec(
284             parentWidthMeasureSpec,
285             childWidthMeasureSpec,
286             childHeightMeasureSpec);
287     }
288     @Override
289     protected void measureChildWithMargins(View child,
290         int widthUsed,
291         int heightUsed,
292         int rightMargin,
293         int bottomMargin,
294         int leftMargin) {
295         MarginLayoutParams lp =
296             MarginLayoutParams(chl.getLayoutParams());
297         lp.widthMeasureSpec = MeasureSpec.makeMeasureSpec(
298             lp.leftMargin + lp.width);
299     }
300 }

```

```

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 */

```

```

387     * / the line .
388     public void scrollToTextChange() {
389         //add line numbers width to get total width
390         Point position = editText.getCursorPosition();
391         position.x = position.x + lineNumbers.getWidth();
392         Point center = getScreenCenter();
393         int scrollX = getScrollX();
394         int scrollY = getScrollY();
395         boolean visible = new Rect(scrollX, scrollY, scrollWidth() + scrollX, getHeight() + scrollY) {
396             .contains(position.x,
397             position.y);
398         );
399         if (!visible) {
400             int x = position.x - center.x;
401             int y = position.y - center.y;
402             scrollTo(x, y);
403             invalidate();
404         }
405         private Point getScreenCenter() {
406             return new Point(getWidth() / 2, getHeight() / 2);
407         }
408     }
409 }
410 
```

Listing B.29: RowEditorLinearLayout.java

```

1   package de.muffinworks.knittingapp;
2   import android.app.Activity;
3   import android.content.Intent;
4   import android.os.Bundle;
5   import android.view.Menu;
6   import android.view.MenuItem;
7   import android.view.View;
8   import android.widget.FrameLayout;
9   import android.widget.ImageButton;
10  import android.widget.TextView;
11  import android.widget.Toast;
12  import java.io.IOException;
13  import java.io.IOException;
14  import de.muffinworks.knittingapp.layouts.*;
15  import de.muffinworks.knittingapp.models.Pattern;
16  import de.muffinworks.knittingapp.storage.util.Constants;
17  import de.muffinworks.knittingapp.views.PatternGridView;
18  import de.muffinworks.knittingapp.views.RowEditorLinearLayout;
19  import de.muffinworks.knittingapp.views.RowEditorRowPattern;
20  public class ViewerActivity extends BaseActivity {
21      private int mCurrentRow = 1;
22      private TextView mRowText;
23      private FrameLayout mPatternContainer;
24      private PatternGridView mGridPattern;
25      private RowEditorLinearLayout mRowPattern;
26      private boolean mIsRowFormatActive = false;
27      private Pattern mPattern;
28      @Override
29      protected void onCreate(Bundle savedInstanceState) {
30          super.onCreate(savedInstanceState);
31          setContentView(R.layout.activity_viewer);
32          enableActionBar(true);
33          protected void onCreate(Bundle savedInstanceState) {
34              super.onCreate(savedInstanceState);
35              setContentView(R.layout.activity_viewer);
36          }
37          String patternId = getIntent().getStringExtra(
38              Constants.EXTRA_PATTERN_ID);
39          if (patternId != null) {
40              mPattern = mStorage.load(patternId);
41              setActionBarTitle(mPattern.getName());
42          }
43          initEditors();
44          initCounter();
45      }
46      @Override
47      public boolean onOptionsItemSelected(MenuItem item) {
48          int id = item.getItemId();
49          if (id == R.id.resetRowCounter) {
50              updateRowCounter();
51          }
52          mGridPattern.scrollCurrentRowToCenter();
53          return true;
54          else if (id == R.id.switch_view_style) {
55              switchEditors();
56          }
57          else if (id == R.id.scrollToCurrentRowToCenter) {
58              mGridPattern.scrollToCurrentRowToCenter();
59          }
60          Intent intent = new Intent(this,
61              EditorActivity.class);
62          intent.putExtra(Constants.EXTRA_PATTERN_ID,
63              mPattern.getId());
64          startActivityForResult(intent, Constants.
65              REQUEST_CODE_EDITOR);
66          Intent intent = new Intent(this,
67              GlossaryActivity.class);
68          intent.putExtra(Constants.EXTRA_PATTERN_ID,
69              exportPattern());
70      }
71      private void exportPattern() {
72          try {
73              mStorage.export(mPattern.getId());
74              showAlertDialog(getString(R.string.
75                  setActionBarTitle(mPattern.getName())));
76          }
77      }
78      return super.onOptionsItemSelected(item);
79  }
80 
```

```

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
```

Listing B.32: activity_glossary.xml

```

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

Listing B.33: activity_pattern_list.xml

```

12 <?xml version="1.0" encoding="utf-8"?>
13 <LinearLayout
14   xmlns:android="http://schemas.android.com/apk/
15     res/android"
16   xmlns:app="http://schemas.android.com/apk/res-auto"
17   android:orientation="vertical"
18   android:layout_width="match_parent"
19   android:layout_height="match_parent"
20   android:layout_margin="11px" />
21 <FrameLayout
22   android:id="@+id/editor"
23   android:textSize="80sp"
24   android:layout_width="match_parent"
```

APPENDIX B. SOURCE CODE

li

```

24      android:id="@+id/containerControls"
25      android:layout_height="0dp"
26      android:orientation="horizontal"
27      android:layout_weight="3">
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          <LinearLayout
37              android:layout_width="wrap_content"
38              android:layout_height="wrap_content"
39              android:layout_gravity="top|center_horizontal"
40              android:layout_weight="1">
41                  <TextView
42                      android:textColor="@color/offblack"
43                      android:textSize="20sp"
44                      android:text="Reihe"/>
45
46                  <TextView
47                      android:id="@+id/row"
48                      android:layout_width="wrap_content"
49                      android:layout_height="wrap_content"
50                      android:layout_gravity="bottom|center_horizontal"
51                      android:gravity="center"
52                      android:textColor="@color/offblack"
53                      android:textSize="70sp"
54                      android:text="120"/>
55
56      </LinearLayout
57      <LinearLayout
58          android:id="@+id/containerControls"
59          android:orientation="horizontal"
60          android:gravity="center"
61          android:layout_weight="3">
62          <ImageButton
63              android:id="@+id/button_increase"
64              android:elevation="4dp"
65              android:layout_width="0dp"
66              android:layout_height="match_parent"
67              android:background="@color/colorAccent"/>
68
69          <ImageButton
70              android:id="@+id/button_decrease"
71              android:layout_marginLeft="7dp"
72              android:elevation="4dp"
73              android:layout_width="75dp"
74              android:layout_height="75dp"/>
75
76          <ImageButton
77              android:background="@drawable/ripple_round"
78              android:src="@drawable/ic_remove"/>
79
80          <ImageButton
81              android:background="@drawable/ic_remove_white"/>
82
83      </LinearLayout>
84
85      </LinearLayout>
86
87  </LinearLayout>

```

Listing B.34: activity-viewer.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

```

Listing B.35: attr.xml

```

18      name="primaryText">#212121</color>
19      name="secondaryText">#727272</color>
20      name="divider">#B6B6B6</color>
21
22      name="offWhite">#dedede</color>
23      name="black-10">#33a1a1</color>
24      name="black-100">#335555</color>
25      name="black-30">#332929</color>
26      name="black-40">#665577</color>
27      name="black-60">#995577</color>
28      name="black-80">#cc5577</color>
29      name="offBlack">#2f2f2f</color>
30
31      name="red_400">#EF5350</color>
32
33  </resources>

```

Listing B.36: colors.xml

```

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:gravity="center"
6         android:weightSum="2"
7           android:layout_width="match_parent"
8             android:layout_height="match_parent">
9
10    <LinearLayout
11      android:orientation="horizontal"
12        android:layout_width="match_parent"
13          android:height="0dp"
14            android:gravity="center"
15              android:layout_weight="1">
16
17    <TextView
18      android:textSize="20sp"
19        android:layout_marginRight="15dp"
20          android:id="@+id/edittext_columns"
21            android:text="String/Columns"
22              android:gravity="right"
23                android:inputType="number"
24                  android:digits="1234567890"
25                    android:layout_width="wrap_content"
26                      android:id="@+id/edittext_columns"
27                        android:layout_height="wrap_content" />
28
29    <EditText
30      android:id="@+id/edittext_actionNext"
31        android:inputType="number"
32          android:digits="1234567890"
33            android:layout_width="100dp" />

```

```

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

```

```

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

```

Listing B.37: dialog_set_grid_size.xml

4 </resources>

Listing B.38: dimens.xml

```

32   android:layout_height="50dp" />
33 </LinearLayout>
34 </LinearLayout>
35 <LinearLayout
36   android:orientation="horizontal"
37     android:layout_width="match_parent"
38       android:layout_height="0dp"
39         android:gravity="center"
40           android:layout_weight="1">
41
42    <TextView
43      android:textSize="20sp"
44        android:id="@+id/edittext_rows"
45          android:imeOptions="actionDone"
46            android:inputType="number"
47              android:gravity="right"
48                android:layout_width="wrap_content"
49                  android:layout_height="wrap_content" />
50
51    <EditText
52      android:id="@+id/edittext_actionDone"
53        android:layout_marginRight="15dp"
54          android:id="@+id/edittext_number"
55            android:digits="1234567890"
56              android:layout_width="100dp"
57                android:layout_height="50dp" />
58
59 </LinearLayout>
60 </LinearLayout>

```

```

1 <resources>
2   <dimen name="actionDone">3dp</dimen>
3   <dimen name="colorPrimary">@color/colorPrimary</dimen>

```

```

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="match_parent"
22         android:layout_height="match_parent"
23           android:layout_weight="1"
24             android:layout_weight="1"
25               android:layout_weight="1"
26                 android:weightSum="3" />
27

```

Listing B.37: dialog_set_grid_size.xml
Listing B.38: dimens.xml
Listing B.39: dimens-w820.xml

```

1 <resources>
2   <dimen name="activity_horizontal_margin">16dp</dimen>
3 </resources>

```

```

1 <resources>
2   <dimen name="row_editor_default_text_size">50sp</dimen>
3 </resources>

```

```

28 <GridView
29   android:id="@+id/keyboard_gridview"
30   android:background="@color/colorAccent"
31   android:fadeScrollbars="false"
32   android:layout_width="0dp"
33   android:scrollbarSize="10dp"
34   android:layout_weight="6"/>
35   android:layout_height="wrap_content"
36   android:horizontalSpacing="20dp"
37   android:horizontalColumns="4"/>
38 </LinearLayout>
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   <ImageButton
46     android:src="@drawable/ic_delete_white_48dp"
47     android:layout_width="match_parent"
48     android:layout_height="match_parent"
49     android:background="@drawable/
50       keyboard_button_background"
51     android:onClick="onDeleteToggled" />
52 </LinearLayout>
53 </LinearLayout>
54 </LinearLayout>
55 </LinearLayout>
56 </LinearLayout>
57 </LinearLayout>
58 <ImageButton
59   android:layout_width="0dp"
60   android:layout_height="3" />
61 <LinearLayout
62   android:orientation="vertical"
63   android:background="@color/colorPrimary"
64   android:layout_width="0dp"
65   android:layout_height="2" />
66 <ImageButton
67   android:layout_width="match_parent"
68   android:layout_height="2" />
69 <ImageButton
70   android:layout_width="0dp"
71   android:layout_height="3" />
72 <ImageButton
73   android:layout_width="0dp"
74   android:layout_height="3" />
75 <ImageButton
76   android:layout_width="0dp"
77   android:layout_height="3" />
78 </ImageButton>
79 </ImageButton>
80 </ImageButton>
81 </ImageButton>
82 </ImageButton>
83 <FrameLayout
84   android:background="@color/colorAccentDark"
85   android:layout_width="3dp"
86   android:layout_height="match_parent" />
87 <include
88   layout="@layout/view_numpad" />
89 </FrameLayout>
90 <include
91   layout="@layout/editor_row.xml" />
92 </FrameLayout>
93 </FrameLayout>

```

Listing B.40: fragment_editor_grid.xml

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_title_grid_size">Change grid size</
9     string>
10    <string name="dialog_title_pattern_delete">Delete
11    pattern %1$s?</string>
12    <string name="dialog_title_pattern_name">Pattern name</
13    string>
14    <!--Activity titles-->
15    <string name="activity-title-glossary">Glossary</string>
16    <!--errors-->
17    <string name="error_over_max_size">Only %1$s supported</
18    string>
19    <string name="error_must_implement_interface">
20    translatable=false> must implement %$s</string>
21    <string name="error_load_metadata">Could not close file
22    reader</string>
23    <string name="error_update_metadata">Could not write
24    metadata</string>
25    <string name="error_server_pattern">Could not write
26    pattern to disk</string>
27    <string name="error_file_not_found">Could not find file:
28    %1$s</string>
29    <string name="error_dimension_zero">Must be at least 1</
30    string>
31    <string name="error_external_storage_not_writable">
32    External storage is not writable</string>
33    <string name="error_external_storage_not_mounted">
34    External storage not available</string>
35    <string name="error_export_failed">Export failed</string>
36    <string name="error_import_no_json">Import failed: can \
37    t read file</string>
38    <!--success-->
39    <string name="success_export_pattern">Successfully
40    exported pattern to folder %1$s on SD card</string>
41    <string name="success_export_all">Successfully exported
42    to folder %1$s on SD card</string>
43    <string name="success_save_pattern">Saving successful</
44    string>
45    <!--menu-->
46    <string name="menu_grid_size">Grid size</string>
47    <string name="menu_switch_editor">Switch view</string>
48    <string name="menu_save">Save</string>
49    <string name="menu_edit_name">Change pattern name</
50    string>
51    <string name="menu_delete_pattern">Delete pattern</
52    string>
53    <string name="menu_reset_counter">Reset counter</string>
54    <string name="menu_open_glossary">Open glossary</string>
55    <string name="menu_open_editor">Edit pattern</string>
56    <string name="menu_jump_to_current_row">Current row</
57    string>
58    <string name="menu_export_all">Export all</string>
59    <string name="menu_export_pattern">Export pattern</
60    string>
61    <string name="rows">rows</string>
62    <string name="placeholder_pattern_name">Pattern name</
63    string>
64    <string name="placeholder_line_numbers">translatable =
false</string>
65    </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 <?xml version="1.0" encoding="utf-8"?>
11 <LinearLayout
12   xmlns:android="http://schemas.android.com/apk/res/
13   android:orientation="horizontal"
14   android:layout_width="match_parent"
15   android:layout_height="match_parent">
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)