

MOBILE DISPLAY OF KNITTING PATTERNS

BACHELOR'S THESIS

BIANCA PLOCH

540609

16 AUGUST 2016

SUPERVISOR:

PROF. DR. DEBORA WEBER-WULFF

HTW BERLIN

INTERNATIONAL MEDIA AND COMPUTING (BACHELOR)

Contents

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
3 Requirements	11
3.1 Functional Requirements	11
3.2 Non-functional Requirements	13
4 Design	14
5 Android Basics	17
5.1 The Operating System Android	17
5.2 Basic Components of an Android App	17
5.2.1 Activity	17
5.2.2 Actionbar	19

5.2.3	Fragment	19
5.2.4	View	21
5.2.5	Storage	22
6	Implementation	23
6.1	Stitch symbols	23
6.2	Pattern and Parsing between Pattern Formats	24
6.3	Persistent Disk Storage	25
6.4	Displaying a Pattern	26
6.4.1	Grid Format	26
6.4.2	Row Format	29
6.5	Keyboard	31
6.6	Viewer with Row Counter	32
6.7	Editor	33
Editor Fragments	34	
6.8	Pattern List	34
6.9	Glossary	35
7	User Test	36
8	Evaluation and Discussion	39
9	Outlook	41
Abbreviations		42
List of Figures		43
Listings		45
Bibliography		46
A User Interviews		i
A.1	Question catalogue	ii
A.2	Interview with Thilo Ilg	ii
A.3	Interview with Nadine Kost	iii
A.4	Interview with Angela Thomas	iv

CONTENTS

iii

B Source Code

vi

Chapter 1

Introduction

Since the beginning of the 2000s, knitting has encountered a steady rise in popularity, claims an article by Lewis 2011. This, she says, might be due to the rise of the internet and social media, and the increasingly important role they play in the daily life. The older knitting generation is adapting to new technology and switching over, Nielsen explains, bringing knitting as a craft and hobby closer to the younger generations (Lewis 2011). Online communities like Ravelry¹ and Youtube² teach the knitting enthusiasts knittings techniques and patterns of all kinds — never has knitting knowledge been more accessible.

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

¹<http://www.ravelry.com/>

²<https://www.youtube.com/>

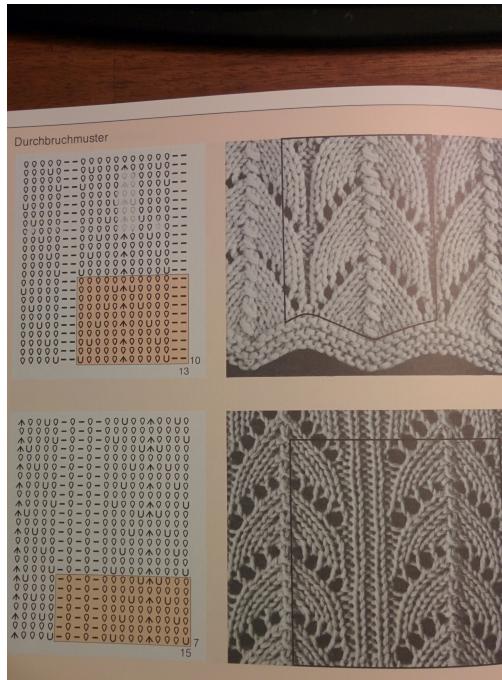


Figure 1.1: Knitting patterns and their corresponding pattern charts from Natter 1983, p142

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

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

Chapter 2

Background

2.1 Definition of Knitting Pattern and Knitting Pattern Chart

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

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

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

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

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

2.2 Comparison of Existing Solutions

2.2.1 Android apps

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

knit tink — Row Counter by Jennifer K. Warren

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

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

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

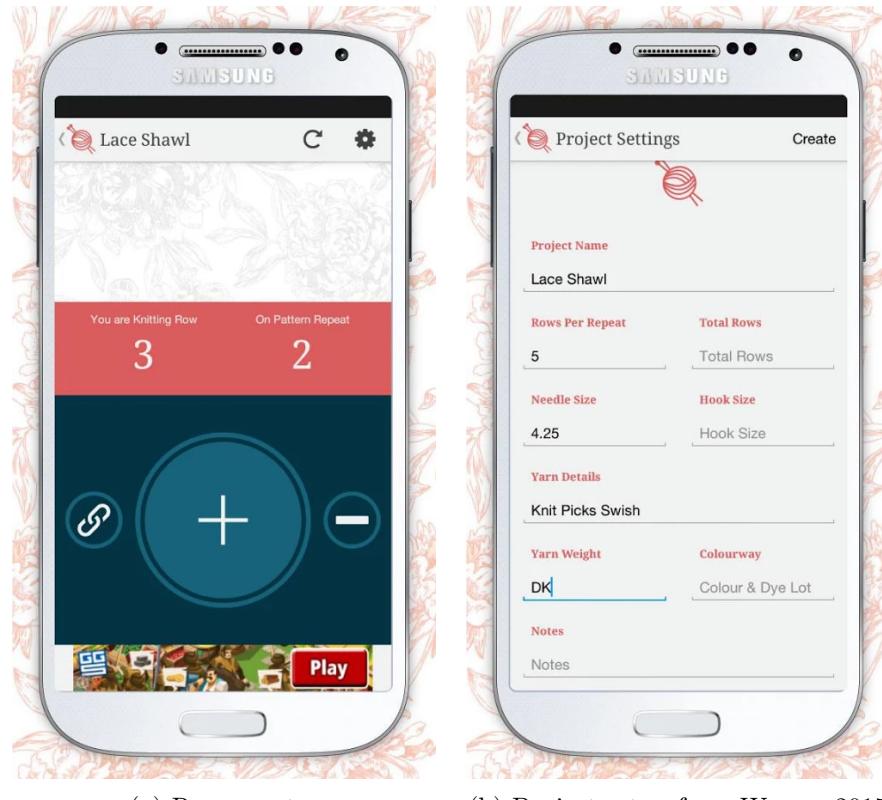
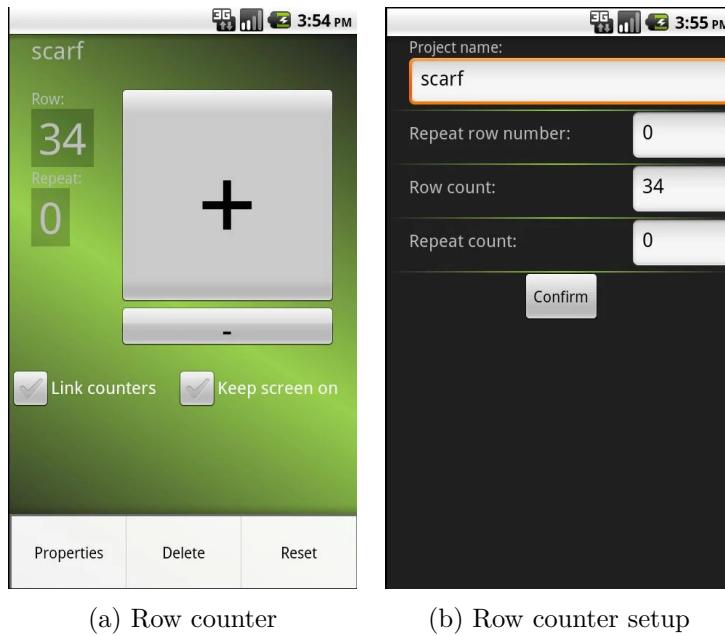


Figure 2.1: Screenshots of the app knit tink

Knitting Counter by mkacki

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

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



(a) Row counter

(b) Row counter setup

Figure 2.2: Screenshots of the app Knitting Counter

Knitting and Crochet Buddy by Colorwork Apps

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

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

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

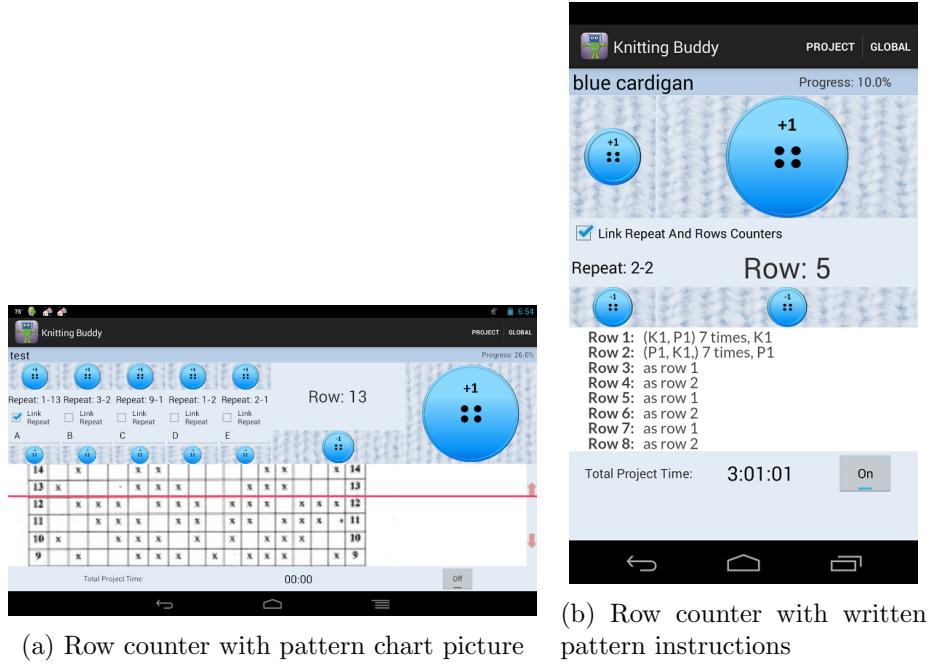
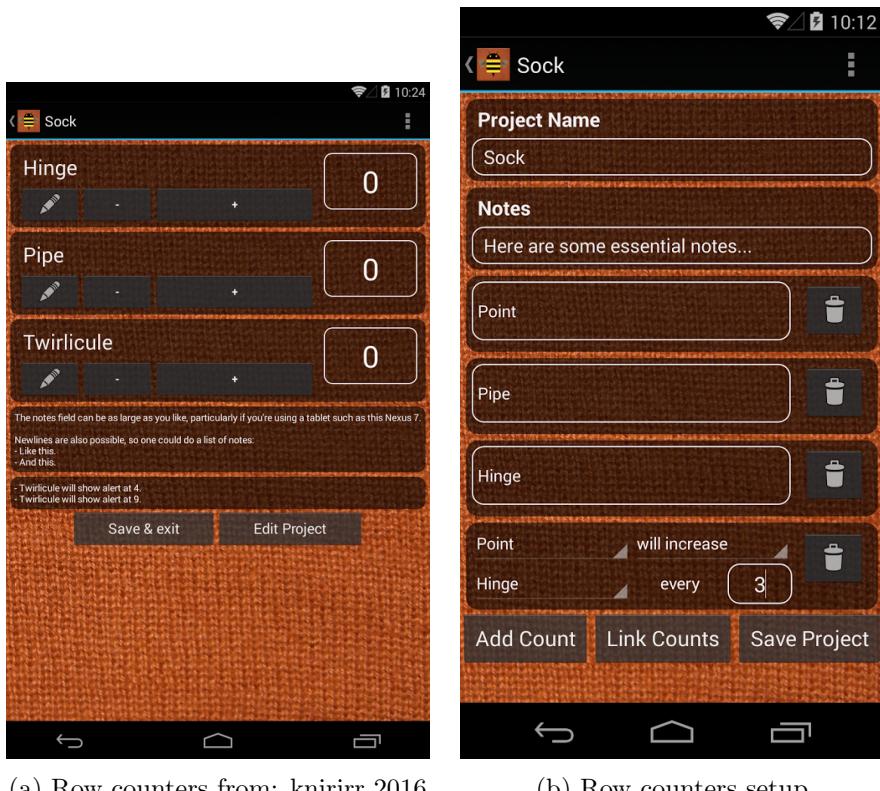


Figure 2.3: Screenshots of the app Knitting and Crochet Buddy

BeeCount knitting Counter by knirirr

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

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



(a) Row counters from: knirrr 2016

(b) Row counters setup

Figure 2.4: Screenshots of the app BeeCount Knitting Counter

Knitting Chart Maker by Awesome Applications

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

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

The app has over 30 stitch symbols that the user can use to create a pattern chart. The symbols are defined by the app and are not taken from a standard. The user cannot devise their own stitch symbols. Symbols can be used by selecting the symbol in the left-hand menu and then transferred onto the grid by tapping on a cell. Alternatively, the user can select the paintbrush button on the top-left menu and use their finger to

paint the symbols onto every cell touched in a swiping motion, not unlike drawing with a pencil. The whole grid is zoomable up to a certain zoom level.

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

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

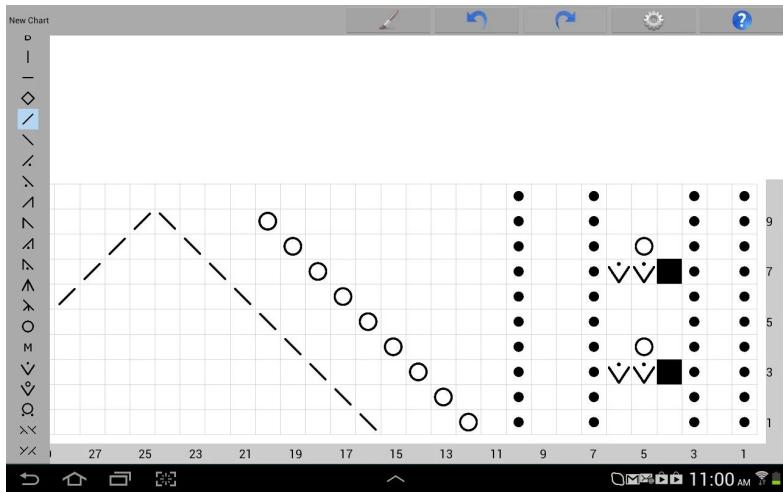


Figure 2.5: Chart editor of Knitting Chart Maker

2.2.2 Other

KnitML by Jonathan Whitall

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

KnitML has no connection to Android, but has an honorable mention here since it addresses the problem of inputting a knitting pattern. KnitML is an XML based format for describing the knitting process from beginning to the finished product. With KnitML

the project aims to establish an international standard for knitting pattern expression¹. He aims to do so by using the Knitting Expression Langauge, KEL, that he defined (Whitall 2009). KEL is based on the Groovy programming language² for the Java platform and the GroovyMarkup architecture.

The following KEL expression

```

1 Pattern {
2     generalInformation
3 }
```

Listing 2.1: Example expression in KnitML

would result in

```

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

Listing 2.2: Example expression in KnitM: XML result

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

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

²<http://groovy-lang.org/templating.html>

Chapter 3

Requirements

3.1 Functional Requirements

The requirements for this thesis are formed from interviews conducted with volunteers at the beginning of this thesis. Three participants, stemming from both the author's acquaintances as well as from volunteers recruited from a poster posted publicly nearby the HTW's campuses, have been interviewed. Prerequisite for a participant in such an interview was a proficiency and an interest in knitting. During a time frame of 45 to 60 minutes the participants were asked to answer a set of questions concerning their knitting experience as well as what features they would like to see in an app aimed to aid them during the creation and viewing of a knitting pattern chart. The catalogue of the questions asked and the answers given by the participants during these interviews can be found in the appendix A.4. From these interviews user stories were formulated and corresponding functional requirements were extracted — see table 3.1 below.

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

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

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

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

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

3.2 Non-functional Requirements

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

Chapter 4

Design

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

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

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

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

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

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

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

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

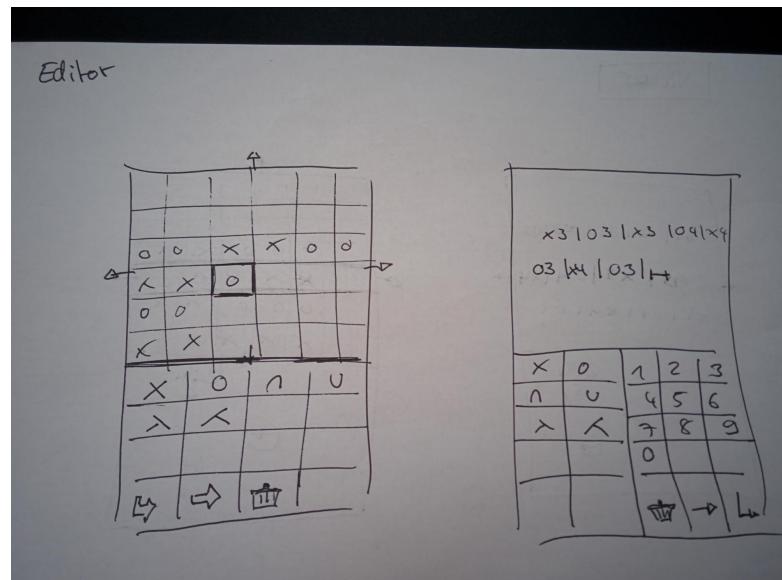


Figure 4.1: Editor screens for grid and row format

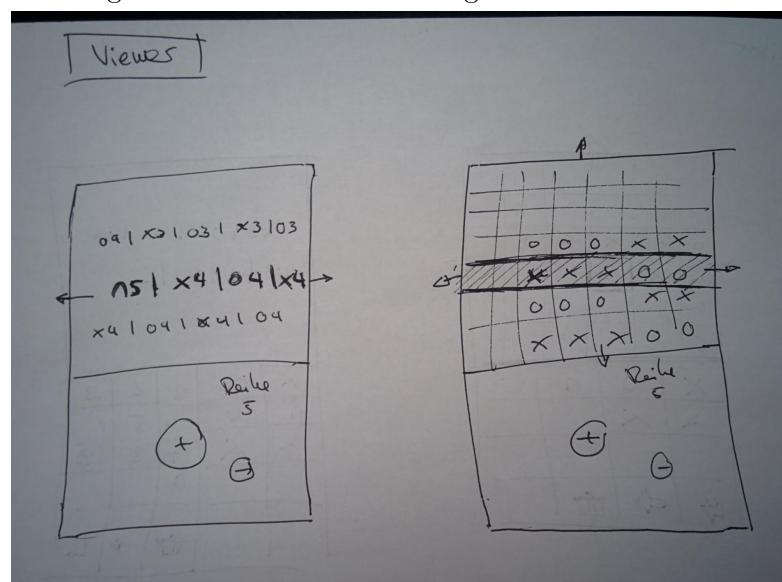


Figure 4.2: Viewer screens for grid and editor format with row counter

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

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

Chapter 5

Android Basics

5.1 The Operating System Android

Android is an open source operating system for mobile phones and tablets based on the Linux kernel (Android Developers 2016d). Android apps are distributed on Google Play, a service owned by Google. To build an Android app Google offers the Android Software Development Kit (SDK), containing sample projects, necessary Android libraries and an Android emulator. Additionally, Google recommends to use the official Android Integrated Development Environment (IDE) Android Studio, which is based on IntelliJ IDEA and offers many useful tools, including testing frameworks, a Graphical User Interface (GUI) for screen layouts, and the build tool Gradle (Android Developers 2016j). The concepts and Android components discussed throughout this chapter are taken from the Android Developer reference¹, training², and Application Programming Interface (API) guides³ found online.

5.2 Basic Components of an Android App

5.2.1 Activity

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

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

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

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

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

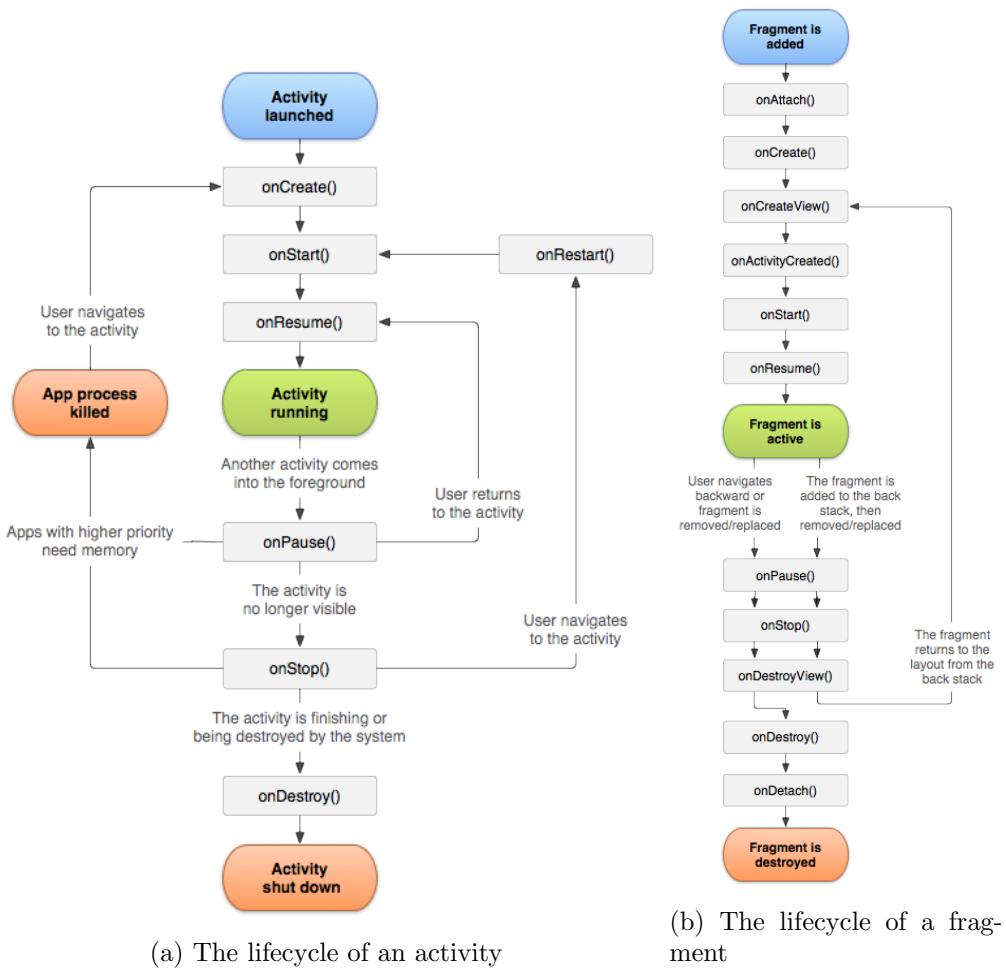


Figure 5.1: The lifecycles of activities and fragments

5.2.2 Actionbar

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

5.2.3 Fragment

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

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

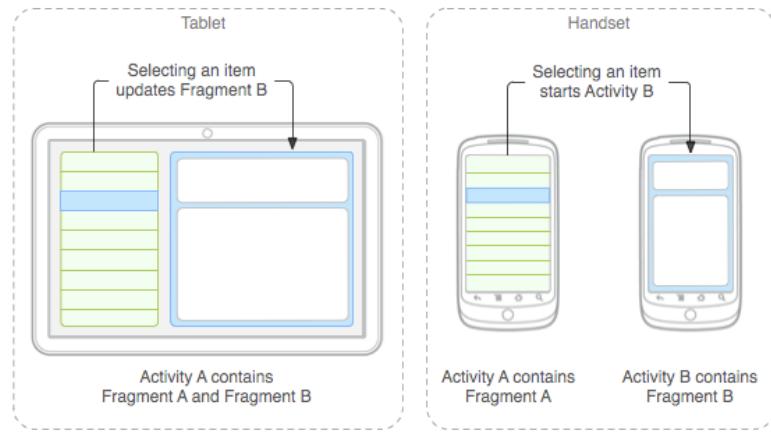


Figure 5.2: Two fragments of one activity and their layout on two different screen sizes

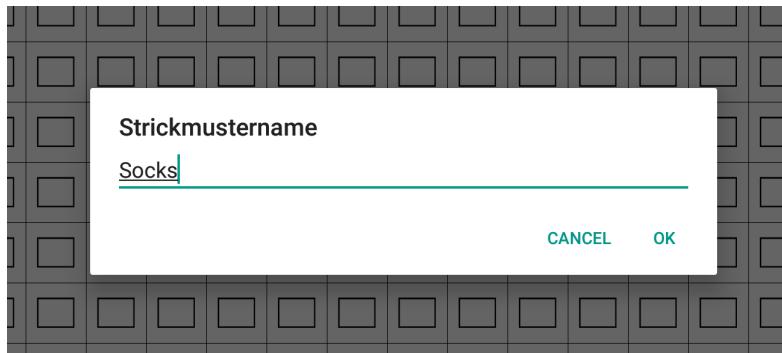


Figure 5.3: A dialog fragment for naming a pattern

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

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

```

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

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

5.2.4 View

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

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

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

5.2.5 Storage

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

Chapter 6

Implementation

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

6.1 Stitch symbols

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

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

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

The open source program FontForge¹ was used creation of the custom TTF knitting font used in the prototype. The knitting font contains a selection of 16 stitch symbols whose glyphs were defined by the author herself. The glyphs and their corresponding UTF-8 characters can seen in **Figure 6.1**. The available symbols the knitting font offers as well as the corresponding symbol descriptions need to be defined as string arrays in the Constants class in the project.

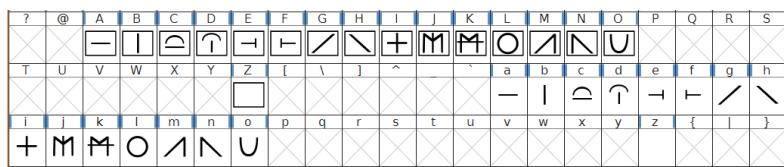


Figure 6.1: The custom knitting font used in the prototype

6.2 Pattern and Parsing between Pattern Formats

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

¹<https://fontforge.github.io/en-US/> (last accessed: 08-10-2016)

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

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

6.3 Persistent Disk Storage

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

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

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

`PatternStorage.java` (see B.24).

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

6.4 Displaying a Pattern

6.4.1 Grid Format

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

Android's code class, on the other hand, is designed with notion of displaying data. Each cell represents one data entry in a collection of data, the displaying of which is handled by an Adapter class attached to the grid view. Android ships with a specialized adapter for lists³, as well as a `BaseAdapter` that can be subclassed for a custom handling and presentation of data. While this is a fitting solution for displaying symbols in a grid, it does not meet the requirements this project sets for the grid format. It is required that the column and row numbers are displayed next to the grid as axes. These axes should scale and scroll together with the grid, but should not be scrolled outside the

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

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

Therefore, in order to fulfill all requirements it makes the most sense to create a custom implementation of a view that supports the display of text in a grid, two-dimensional scroll, zoom, and axes that stick to the view bounds. Google's sample project *Interactive Chart*⁵ and the corresponding training path⁶ present an implementation example and were used as a guideline for the implementation of the class `PatternGridView.java` (see B.18). This class keeps a reference to a two-dimensional array of strings which represents the pattern with its columns and rows. The grid and its axes are drawn by overriding the `onDraw()` method, calculating the position of the corresponding lines and numbers with the dimensions of the view and pre-defined, hardcoded values for cell width and margin. The current version of the `PatternGridView` uses a `SimpleOnGestureListener`⁷ to compute two-dimensional dragging. Android defines two different scrolling types for views: dragging and flinging (Android Developers 2016e). Dragging is executed by dragging a finger across the device's touchscreen and results in a moving of the view corresponding to the dragging direction and speed. This means, that no matter the velocity of the dragging gesture, the view will not keep moving once the user lifts the finger involved in the gesture. A fling will keep moving the view with a speed and duration exponential to the velocity of the fling gesture, where the duration is calculated by introducing a

⁴<https://developer.android.com/reference/android/webkit/WebView.html> (accessed: 11-08-2016)

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

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

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

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

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

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

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

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

Similarly to dragging, scaling is implemented with a `SimpleOnScaleGestureListener`⁹ with is connected to the view's `onTouchEvent()`. The scaling factor is then clamped at pre-defined maximum and minimum values and saved in a variable. During the drawing operations of the grid, axes, and text the scaling factor is then used to compute and draw the scaled visuals. On user touch on the grid the touched cell is calculated from the pixel position of the touch event and the currently selected stitch string is saved to that position in the two-dimensional pattern array. Following this the view is invalidated¹⁰ and re-drawn with the updated pattern.

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

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

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

6.4.2 Row Format

The row editor should display line numbers at the left side of the screen and keep them in that position during horizontal scrolling, so that they will not move offscreen. Additionally, it should support the standard text editor functions: text select, copy, cut, and paste. It should display text in multiple lines that do not wrap at the end of the screen, but continue offscreen until a newline is input and the content needs to be scrollable horizontally and vertically. Android's `Edittext` widget¹¹ fulfills most of these requirements. The `Edittext` widget inherits from the class `EditText` which is specialised for displaying text content. It supports standard text editing functions, can display multiple lines of text, and supports vertical scrolling. Unfortunately, it does not natively support text lines to continue offscreen — upon reaching the width of the widget the text is wrapped to the next line. Another problem is, that the widget always automatically triggers the showing of Android's on-screen keyboard when it receives focus, i.e. when the user taps the view to start text input.

One instance, when the on-screen keyboard, also called the soft input method (Android Developers 2016i), is shown, is when the activity's main window has input focus (Android Developers 2016c). An activity receives input focus on activity start and resume when containing an `Edittext` widget in its view hierarchy. This behavior can be suppressed by declaring the state of the soft input method in the manifest file (see B.1) of the project as hidden (see *Listing 6.1*).

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

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

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

¹¹<https://developer.android.com/reference/android/widget/EditText.html>

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

To improve the visibility of the lines the `LinedEditor` (see B.13) class is subclassed from the keyboardless `Edittext` widget and a background is drawn behind every other line. The `LinedEditor` is part of the `RowEditorLinearLayout` (see B.27), a custom `LinearLayout` which houses the complete row format editing functionality. This layout will be referred to as the row editor. A `LineNumberTextView` (see B.14), a custom `TextView`, is placed to the left of the `LinedEditor` and displays the line numbers. To achieve a consistent look of both line numbers and editor text the same font and text size are set on both `EditText` and `Edittext` widget. To suppress the `Edittext` widget's line wrap behavior its width is set to the value `wrap_content` (see *Listing 6.2*). This allows the widget to increase its width when text is added that exceeds the current width of the view.

```

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

```

Listing 6.2: Excerpt from `row_editor.xml`

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

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

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

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

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

The `Edittext` widget also returns the wrong position when more text is added at the end of a line. Instead of calculating the cursor's new position by increasing the value of its x-coordinate as by the width of added text, the returned position is at the first character of the next line. This might be due to the suppressed line wrapping behavior of `Edittext` widget in multi-line mode. Despite returning an incorrect cursor position upon text change, the text and cursor are 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. This behavior might be achieved by attaching different scrollers to the `EditText` for the line numbers and the `Edittext` widget, but the time constraints of this thesis did not allow further research.

6.5 Keyboard

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

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

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

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

The symbols sections is also a `Gridview`, although with less columns than in the grid editor. The numpad uses the `CalculatorPadLayout` from Rahul Parsani's Material Calculator project¹⁵. The `CalculatorPadLayout` takes a number of child views, in this case `KnittingFontButtons`, as well as arguments for row and column count. It then calculates the size of the child views, so that all are equal in size. This custom layout is used because it optimally arranges its children in the available space without scrolling. A `Gridview`, on the other hand, is built to dynamically accommodate data and possible data changes — it is only concerned with the number of columns the data views can be placed in, if the `Gridview` bounds are too small to display all rows they will automatically placed offscreen and the `Gridview` will become scrollable — an undesired and atypical behaviour for a number pad, in the author's opinion.

6.6 Viewer with Row Counter

The row counter and the viewer are implemented in the `ViewerActivity` class (see B.28). The activity's layout file defines a container `FrameLayout`¹⁶ for the pattern content and below that the row counter UI. On its creation the activity instantiates a `PatternGridView` and a `RowEditorLinearLayout` and sets the data of the viewed pattern on both. The view and the layout can then be switched out at runtime inside their container by adding and removing the required view whenever the user decides to switch

¹⁵<https://github.com/rahulparsani/material-calculator>

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

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

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

The actionbar contains the following action buttons

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

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

6.7 Editor

The class `EditorActivity.java` (see B.4) 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 the user to save the changes or to discard them and close the editor. After a pattern is exported an info dialog is displayed the directory on the external storage that the file was exported to. The activity also handles the showing of dialog fragments to request user input and processes the results. The dialogs handled in the `EditorActivity` are the `GridSizeDialogFragment` (see ??), the `PatternDeleteDialogFragment` (see B.17), and the `PatternNameDialogFragment` (see ??).

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 B.19) that displays all files currently indexed in the `Metadata` file (see section 6.3). For that Android's `ListView`¹⁷ component is used. For each file the pattern name, a button to edit (pencil icon), and a button to delete (trash can icon) the pattern are shown. The edit button opens the selected pattern in the `EditorActivity` and upon delete the `PatternDeleteDialogFragment` is shown.

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

6.9 Glossary

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

¹⁸see footnote 17

Chapter 7

User Test

As stated in Chapter 1 the prototype's UI is evaluated by iterative UI testing. For this a user is given a device running the prototype and a set of tasks related to the features of the prototype. While the user executes these tasks their reactions while using the prototype are observed and instances of problems with the UI noted. The user is then asked to summarize their experience in regards how easily they were able to use the features of the 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. After a brief explanation on what the number and symbol combination signified in the row format and how it would be displayed expanded when viewed in the grid format both participants were quickly able to work with the row editor and accurately produce results they were aiming for. For this they at first relied on switching to the grid editor to see how their changes in the row editor would play out — they were only able to grasp entirely how the row editor functioned after seeing the changes they did to the pattern in the expanded grid format. The participants did not expect the editor to support the standard editor 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.

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

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

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

The participants agreed that the 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.

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

Chapter 8

Evaluation and Discussion

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

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

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

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

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

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

Chapter 9

Outlook

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

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

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

Abbreviations

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

GUI Graphical User Interface. 19

IDE Integrated Development Environment. 19

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

k2tog knit two together. 43

POJO Plain Old Java Object. 26–28

RS right side. 3, 4

SDK Software Development Kit. 19

TTF True Type Font. 26

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

USB Universal Serial Bus. 24

UUID Universally Unique Identifier. 27, 28

WS wrong side. 4

XML Extensible Markup Language. 23, 24

List of Figures

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

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

Listings

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

Bibliography

- Android Developers. *Action Bar*. URL: <https://developer.android.com/design/patterns/actionbar.html> (visited on 08/09/2016).
- *Activities*. URL: <https://developer.android.com/guide/components/activities.html> (visited on 08/09/2016).
 - *jactivity*. URL: <https://developer.android.com/guide/topics/manifest/activity-element.html> (visited on 08/09/2016).
 - *Android, the world's most popular mobile platform*. URL: <https://developer.android.com/about/android.html> (visited on 08/09/2016).
 - *Animating a Scroll Gesture*. URL: <https://developer.android.com/training/gestures/scroll.html#term> (visited on 08/09/2016).
 - *Choose Internal or External Storage*. URL: <https://developer.android.com/training/basics/data-storage/files.html#InternalVsExternalStorage> (visited on 08/09/2016).
 - *Creating event callbacks to the activity*. URL: <https://developer.android.com/guide/components/fragments.html#EventCallbacks> (visited on 08/09/2016).
 - *Fragments*. URL: <https://developer.android.com/guide/components/fragments.html> (visited on 08/09/2016).
 - *Handling Keyboard Input*. URL: <https://developer.android.com/training/keyboard-input/index.html> (visited on 08/09/2016).
 - *Meet Android Studio*. URL: <https://developer.android.com/studio/intro/index.html> (visited on 08/09/2016).
 - *Saving Data*. URL: <https://developer.android.com/training/basics/data-storage/index.html> (visited on 08/09/2016).
 - *Supporting Multiple Screens*. URL: https://developer.android.com/guide/practices/screens_support.html (visited on 08/09/2016).

- Android Developers. *System Permissions*. URL: <https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous> (visited on 08/09/2016).
- . *Table*. URL: <https://developer.android.com/guide/topics/ui/layout/grid.html> (visited on 08/09/2016).
- . *View*. URL: <https://developer.android.com/reference/android/view/View.html> (visited on 08/09/2016).
- Association, Japan Knitting Certificate. *JIS L 0201-1995: Letter symbols for knitting stitch. Standard booklet*. Nov. 1995. URL: <http://www.webstore.jsa.or.jp/webstore/Com/FlowControl.jsp?lang=en&bunsyoId=JIS+L+0201%3A1995&dantaiCd=JIS&status=1&pageNo=0> (visited on 04/04/2016).
- Clark, Matt. *Android Two-Dimensional ScrollView*. June 2, 2010. URL: <https://web.archive.org/web/20110625064025/http://blog.gorges.us/2010/06/android-two-dimensional-scrollview> (visited on 08/09/2016).
- Kauri. *Kauri's Knitting Font*. July 31, 2016. URL: <https://sites.google.com/site/kauriknitsfont/> (visited on 08/09/2016).
- knirrr. *BeeCount Knitting Counter*. May 8, 2016. URL: https://lh5.ggpht.com/CaLXmsrgU6JmB1iswLwffjY2eMf0gtt90H41RgHRmGPqro6XCrMdnkawc_TR4nhohYI=h900-rw (visited on 08/09/2016).
- Lewis, Perri. *Pride in the wool: the rise of knitting*. July 6, 2011. URL: <https://www.theguardian.com/lifeandstyle/2011/jul/06/wool-rise-knitting> (visited on 08/09/2016).
- Microsoft Corporation. *Recommendations for OpenType Fonts*. May 1, 2014. URL: <https://www.microsoft.com/typography/otspec/recom.htm> (visited on 08/09/2016).
- Natter, Maria. *Stricken*. Niedernhausen: Falken Verlag, 1983.
- Nielsen, Jakob. *Usability 101: Introduction to Usability*. Jan. 4, 2014. URL: <https://www.nngroup.com/articles/usability-101-introduction-to-usability/> (visited on 08/09/2016).
- Raz, Samuel. *Flat Knitting Technology*. Heidenheim: Druck Repo Verlag, 1993.
- Warren, Jennifer K. *Knit tink— Row Counter*. Sept. 16, 2015. URL: <https://lh5.ggpht.com/FfgM0tcw2WerHBjS1eqm8NsjuxnTcfPHvxenf-3hfC1NKsIGHp-SPhcQy07Zpru8AQ=h900-rw> (visited on 04/04/2016).

- Whitall, Jonathan. *The KnitML User's Guide*. Apr. 25, 2009. URL: <http://www.knitml.com/docs/users-guide.html#d0e246> (visited on 08/09/2016).
- Xenakis, David. *Knitter's Symbols Fonts*. 1998. URL: <http://www.knittinguniverse.com/downloads/KFont/> (visited on 08/09/2016).

Appendix A

User Interviews

A.1 Question catalogue

- Q1. *Wie viele Stücke haben Sie im vergangenen Jahr gestrickt?*
- Q2. *Für wen stricken Sie? Enkel, für sich selber?*
- Q3. *Gibt es bestimmte Stücke, die Sie häufig Stricken?*
- Q4. *Benutzen Sie bestimmte Techniken häufiger als andere?*
- Q5. *Wie wählen Sie eine Strickmuster? Selber ausdenken, suchen (online))*
- Q6. *Wie arbeiten Sie damit?*
- Q7. *Wie gehen Sie vor wenn Sie mit einer Strickmusterschematik arbeiten?*
- Q8. *Könnten Sie sich vorstellen ein Strickmuster von einem mobilen Gerät abzulesen?*
- Q9. *In welchem Format würden Sie dies gerne sehen? (audial, visuell)*
- Q10. *Wie würden Sie dem Gerät zu erkennen geben, dass eine Reihe fertig gestrickt wurde? (Sprachbefehl, Knopf drücken)*
- Q11. *(Verschiedene Strickmustertemplates zeigen und nach der Lesbarkeit fragen)*
- Q12. *Haben Sie schon einmal selber Strickmusterschematiken erstellt?*
- Q13. *Könnten Sie sich vorstellen, dies auf einem Handy zu tun?*
- Q14. *Wie würden Sie sich dabei die Eingabe vorstellen?*
- Q15. *Bei welchen Aspekten des Stricken könnten Sie sich eine App als hilfreiche Unterstützung vorstellen*

A.2 Interview with Thilo Ilg

- A1. Hat das letzte mal 2009 gestrickt.
- A2. Hat nur für Schule gestrickt, hatte 6 Jahre lang einen Strickkurs.
- A3. Socken.
- A4. Nadelspiel.

- A5. Von Lehrkraft ausgesucht.
- A6. Hat noch nicht mit Musterschematik gearbeitet.
- A7. -
- A8. Ja.
- A9. Beides ok, bevorzugt visuell.
- A10. Findet Spracheingabe sehr nützlich, Hände sind voll beim Stricken.
- A11. Beide Ansichten sind gut, würde sich aber gestört fühlen bei breiten Mustern vom ständigen Scrollen und würde Landscape-Modus besser finden, da mehr Platz zum Lesen der Reihe.
- A12. Nein.
- A13. Ja.
- A14. Würde gerne Bereiche markieren können für eine Masche. Hätte gerne Funktrion um mehrere Zellen zu markieren und dann mit einem Maschensymbol zu befüllen. Klicken zum auswählen einer Zelle, zB. zum Bearbeiten. wenn Zelle markiert, nach Eingabe eines Symbols soll dann gleich zur nächsten Zelle gesprungen werden.
- A15. Würde gerne Bilder von dem fertigen Gestrickten sehen und schriftliche Anweisungen bevor er sich mit der Schematik befasst. Will für komplexe Muster auf jeden Fall Schematik haben, für simplere Muster eher nicht notwendig. Hätte gerne Symbole in verschiedenen Farben für Sichtbarkeit. Wünscht sich Knopf um auf aktuelle Reihe und default Zoomstufe zu springen.

A.3 Interview with Nadine Kost

- A1. 25.
- A2. Freunde und für den Eigenbedarf.
- A3. Fingerlose Handschuhe, Socken.
- A4. Bevorzugt Rundstricken.

- A5. Internet, würde gerne selber Muster schreiben, arbeitet am häufigsten mit schriftlichen Musteranweisungen.
- A6. Keine besondere Arbeitsweise.
- A7. Ausdrucken und mit einem Stift die vollendeten Reihen durchstreichen.
- A8. Ja.
- A9. Visuell.
- A10. Würde gerne nach der Vollendung einer Reihe einen Knopf drücken können. Dies soll auch ausserhalb der App möglich sein, zum Beispiel wie in Spotify mit einem Eintrag in der Notification bar oder mit einem Lockscreen widget.
- A11. Bevorzugt: Zeilenansicht mit Knopf für den Wechsel zwischen Zeilen- und Zel- lenansicht. Hätte gerne am Ende der Reihe die Anzahl der Maschen angezeigt.
- A12. Nein.
- A13. Ja, kann sich das besonders gut vorstellen für Farbmuster.
- A14. Am Anfang sollte man die Grösse des Musters wählen können. In einem Raster dieser Grösse soll dann bei Tap auf eine Zelle eine Auswahlansicht eingeblendet werden, aus der man Symbole für verschiedene Maschen wählen kann. Nach kurzer Überlegung: es wäre benutzerfreundlicher ein Symbol als aktiv zu kennzeichnen, welches dann bei Klick auf eine Zelle in diese eingetragen wird.
- A15. Beim Reihenzählen in einer Strickmusterschematik. Projektmanagement für Strickprojekte. Als ein Übersetzer von metrischen Einheiten von Nadelgrössen, Gewichten und Längen in imperiale und umgekehrt. Hätte ebenfalls gerne schriftliche Anweisungen in einer App wo man mit Knopfdruck auf nächste An- weisung springen könnte, zB. in Verbindung mit Reihenzähler.

A.4 Interview with Angela Thomas

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

- A2. Größtenteils für Bekannte.
- A3. Stulpen, Dreieckstücher.
- A4. Nein.
- A5. Internet, Strickzeitung, Muster durch Bekannte gelernt.
- A6. Keine Erfahrung mit Musterstricken, hat bisher nur Häkelmuster (Form) benutzt.
- A7. Für Häkelmuster: mit Stecknadel Reihe markieren.
- A8. Ja.
- A9. Hätte gerne eine Sprachausgabe der momentanen Reihe und würde diese dann durch Knopfdruck wieder wiederholen lassen.
- A10. Sprachbefehl: durchaus denkbar.
- A11. Beide Ansichten wurden als wichtig gefunden, ein Wechsel zB per Knopf ist sowohl bei der Mustererstellung als auch in der Strickansicht gewünscht. Zeilenansicht ist für Kurzschrift, Zellen zum genaueren Betrachten des Musters.
- A12. Nein.
- A13. Ja.
- A14. In der Zeilenansicht, wobei dann zwischen Zeilen - und Zellenansicht gewechselt werden kann. Möchte nicht darauf achten zu müssen Zellen zu zählen, daher wird Zeileneingabe bevorzugt.
- A15. Erklärung und anschauliches Beispiel für einzelne Maschen beim Stricken denkbar, Strick-/Häkelmuster auf dem Gerät mitnehmen (hat selber keine Smartphone, könnte sich das aber vorstellen). Bevorzugt schriftliche Anweisungen bei Mustern und braucht Text um eine Musterschematik zu verstehen.

Appendix B

Source Code

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/
3   res/android">
4     <package 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_EXTERNAL_STORAGE) {  
        // should show permission  
        showAlertDialog(getString(R.string.  
            info_storage_permission),  
        new DialogInterface.OnClickListener  
        () {  
            @Override  
            public void onClick(  
                DialogInterface dialog, int  
                which) {  
                requestPermissions(new  
                    String[] { Manifest.  
                        permission.  
                        WRITE_EXTERNAL_STORAGE },  
                    Constants.PERMISSION_REQUEST_WRITE_SD  
                );  
            }  
        });  
        return;  
    }  
    requestPermissions(new String[] { Manifest.  
        permission.WRITE_EXTERNAL_STORAGE },  
        Constants.PERMISSION_REQUEST_WRITE_SD);  
    }  
}  
protected void showAlertDialog(String message) {  
    Dialog.OnClickListener positiveCallback =  
        new DialogInterface.OnClickListener() {  
            @Override  
            public void onClick(DialogInterface dialog, int which) {  
                dialog.dismiss();  
            }  
        };  
    Dialog.OnClickListener negativeCallback =  
        new DialogInterface.OnClickListener() {  
            @Override  
            public void onClick(DialogInterface dialog, int which) {  
                dialog.cancel();  
            }  
        };  
    Dialog dialog = new AlertDialog.Builder(this)  
        .setMessage(message)  
        .setPositiveButton(R.string.dialog_ok,  
            positiveCallback)  
        .setNegativeButton(R.string.dialog_cancel,  
            negativeCallback)  
        .create();  
    dialog.show();  
}
```

Listing B.2: BaseActivity.java

```

1 package de.muffinworks.knittingapp.util;
2
3 import android.os.Environment;
4
5 public final class Constants {
6
7     public static final String EMPTY_SYMBOL = "Z";
8
9     public static String KNITTING_FONT_PATH = "fonts/
10        OwnKnittingFont.ttf";
11
12     public static String METADATAFILENAME = "metadata.
13         json";
14
15     // gets path to external storage that is user
16     // accessible and won't be deleted after app
17     // install
18     public static final String EXPORT_FOLDER_PATH =
19         Environment.getExternalStorageDirectory()
20             .getAbsolutePath()
21             + "/"+ EXPORT_DIR;
22
23     public static int FILE_PICKER_REQUEST_CODE = 2342;
24
25     public static final String EXTRA_PATTERN_ID = "de.
26         muffinworks.EXTRA_PATTERN_ID";
27
28     public static final String EXTRA_PATTERN_DELETED =
29         "de.muffinworks.EXTRA_PATTERN_DELETED";
30
31     public static final int PERMISSION_REQUEST_WRITE_SD
32         = 1337;
33
34     public static final String DEFAULT_PATTERN = {
35         "10": Constants.EMPTY_SYMBOL,
36         "10": Constants.EMPTY_SYMBOL,
37         "10": Constants.EMPTY_SYMBOL,
38         "10": Constants.EMPTY_SYMBOL,
39         "10": Constants.EMPTY_SYMBOL,
40         "10": Constants.EMPTY_SYMBOL
41     };
42
43     public static final String SYMBOL_DESCRIPTIONS =
44         {
45             "Rechte Masche", "Linke Masche", "
46             Rechtsverschränkte Masche", "
47             Linksverschränkte Masche", "
48             Masche rechts abheben", "2 rechts zusammen stricken", "
49             links zusammen stricken", "Randmasche", "
50             1 rechte Masche aufnehmen", "1 linke Masche aufnehmen", "
51             Abbinden", "Links neigende Zunahme", "
52             neigende Zunahme", "
53             Umschlag", "Leerer Platzhalter"
54         };
55
56     public static final int REQUEST_CODE_EDITOR = 1;
57
58     public static final int DEFAULT_ROWS = 10;
59
60     public static final int DEFAULT_COLUMNS = 10;
61
62     public static final int MAX_ROWS_AND_COLUMNS_LIMIT =
63         35;

```

6

Listing B.3: Constants.java

```

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

46   enableActionBar(true);
47   mPatternId = getIntent().getStringExtra(
48     Constants.EXTRA_PATTERN_ID);
49
50   if (mPatternId != null) {
51     mPattern = mStorage.load(mPatternId);
52     setActionBarTitle(mPattern.getName());
53   }
54
55   mRowEditorFragment = RowEditorFragment.*;
56   getINSTANCE(mPatternId);
57   mGridEditorFragment = GridEditorFragment.*;
58   getINSTANCE(mPatternId);
59
60   mFragmentManager = getSupportFragmentManager();
61   FragmentTransaction fm = mFragmentManager.beginTransaction();
62   replace(mFragmentManagerContainer,
63     mGridEditorFragment);
64
65   @Override
66   public boolean onCreateOptionsMenu(Menu menu) {
67     getMenuInflater().inflate(R.menu.menu_editor,
68       menu);
69     mMenuItemSetGridSize = menu.findItem(R.id.set_size);
70     mMenuItemSetGridSize.setOnMenuItemClickListener(
71       true);
72
73   @Override
74   public boolean onOptionsItemSelected(MenuItem item)
75   {
76     int id = item.getItemId();
77     if (id == R.id.set_size) {
78       showSetSizeDialog();
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     } else if (id == R.id.save_pattern) {
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     }
92
93   private FragmentManager mFragmentManager;
94   private RowEditorFragment mRowEditorFragment;
95   private GridEditorFragment mGridEditorFragment;
96   private int mFragmentContainerId;
97   private MenuItem mMenuItemSetGridSize;
98   private Pattern mPattern;
99   private String mPatternId = null;
100  private boolean mWasEdited = false;
101
102  @Override
103  protected void onCreate(@Nullable Bundle
104    savedInstanceState) {
105    super.onCreate(savedInstanceState);
106    setContentView(R.layout.activity_editor);
107  }

```

APPENDIX B. SOURCE CODE

```

90         }
91     } return super.onOptionsItemSelected(item);
92 }
93
94     private void exportPattern() {
95         try {
96             mStorage.export(mPatternId);
97             showAlertDialog(getString(R.string.
98                     success_export_pattern, Constants.
99                     EXPORT_DIR));
100        } catch (IOException e) {
101            e.printStackTrace();
102        }
103    }
104
105    @Override
106    public void onBackPressed() {
107        if (wasPatternEdited()) {
108            AlertDialog saveBeforeExitDialog = new
109                AlertDialog.Builder(this)
110                    .setTitle(getString(R.string.
111                        dialog_title_pattern_save_changes))
112                    .setPositiveButton(getString(R.string.
113                        dialog_positive_button_save),
114                        new DialogInterface.OnClickListener() {
115                            @Override
116                            public void onClick(DialogInterface dialog,
117                                int which) {
118                                dialog.cancel();
119                                setResult(!mWasEdited ? Activity.
120                                    RESULT_CANCELED : Activity.
121                                    RESULT_OK);
122                                finish();
123                            }
124                            create();
125                            saveBeforeExitDialog.show();
126                        } else {
127                            setResult(!mWasEdited ? Activity.RESULT_OK :
128                                RESULT_CANCELED : Activity.RESULT_OK);
129                            super.onBackPressed();
130                        }
131
132        private void switchEditors() {
133            savePattern();
134            FragmentTransaction fm = mFragmentManager.
135                beginTransaction();
136            if (mRowEditorFragment.isVisible())
137                fm.replace(mFragmentManagerContainer,
138                    mGridEditorFragment);
139            mMenuItemSetGridSize.setVisible(true);
140        } else {
141            fm.replace(mFragmentManagerContainer,
142                mRowEditorFragment);
143            mMenuItemSetGridSize.setVisible(false);
144        }
145        fm.commit();
146    }
147    private boolean wasPatternEdited() {
148        if (mRowEditorFragment.isVisible())
149            return mRowEditorFragment.hasPatternChanged
150        else
151            return mGridEditorFragment.hasPatternChanged
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(R.string.
178            tag_dialog_fragment_edit_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        private void showDeletePatternDialog() {
189            PatternDeleteDialogFragment dialog =
190                PatternDeleteDialogFragment.newInstance(
191                    mPattern.getName());
192
193        if (mRowEditorFragment.isVisible())
194            FragmentTransaction fm =
195                beginTransaction();
196
197    }

```

APPENDIX B. SOURCE CODE

```

187     dialog.show(mFragmentManager, getString(R.string
188         .tag_dialog_fragment_delete_pattern));
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     }
208
209     mStorage.save(mPattern);
210     setActionBarTitle(mPattern.getName());
211
212     mWasEdited = true;
213     refreshFragmentData();
214
215     @Override
216     public void onConfirmDelete() {
217         Intent intent = new Intent();
218         intent.putExtra(Constants.EXTRA_PATTERN_DELETED, true);
219         setResult(Activity.RESULT_CANCELED, intent);
220     }
221
222 }
```

Listing B.4: EditorActivity.java

```

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

Listing B.5: GlossaryActivity.java

```

10 import de.muffinworks.knittingapp.R;
11 import de.muffinworks.knittingapp.util.Constants;
12
13 public class GlossaryAdapter extends BaseAdapter {
14     private LayoutInflater mInflater;
15     private Context mContext;
16     private ViewGroup mGroup;
17     private BaseAdapter mAdapter;
18     private TextView mView;
19
20     public GlossaryAdapter(Context context) {
21
22 }
```

```

20     mContext = context;
21     mInflater = (LayoutInflater) mContext
22         .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
23 }
24 @Override
25 public int getCount() {
26     return Constants.SYMBOLS.length;
27 }
28 @Override
29 public Object getItem(int position) {
30     return Constants.SYMBOLS[position];
31 }
32 @Override
33 public long getItemId(int position) {
34     return 0;
35 }
36 @Override
37 public boolean isEnabled(int position) {
38     return false;
39 }
40 @Override
41 public View getView(int position, View convertView,
42     ViewGroup parent) {
43     convertView = LayoutInflater.from(parent)
44         .inflate(R.layout.glossary_item, parent, false);
45     if (convertView == null) {
46         convertView = LayoutInflater.from(parent)
47             .inflate(R.layout.glossary_item, parent, false);
48     }
49     convertView.setLayoutParams(new GlossaryItemViewHolder(
50         convertView).layout);
51     convertView.setTag(new GlossaryItemViewHolder(
52         convertView));
53     convertView.setOnClickListener(new View.OnClickListener() {
54         @Override
55         public void onClick(View v) {
56             String symbol = Constants.SYMBOLS[position];
57             String description = Constants.DESCRIPTIONS[position];
58             String text = Constants.TEXTS[position];
59             String id = String.valueOf(position);
60             String bundleId = String.valueOf(position);
61             String patternId = String.valueOf(position);
62             String bundleDescription = Constants.DESCRIPTIONS[position];
63             String bundleText = Constants.TEXTS[position];
64             String bundleSymbol = Constants.SYMBOLS[position];
65             String bundleIdString = String.valueOf(position);
66             String bundleTextString = Constants.TEXTS[position];
67             String bundleSymbolString = Constants.SYMBOLS[position];
68             String bundleDescriptionString = Constants.DESCRIPTIONS[position];
69             String bundleTextIdString = String.valueOf(position);
70             String bundleSymbolIdString = String.valueOf(position);
71             String bundleDescriptionIdString = String.valueOf(position);
72         }
73     });
74 }
75
76 @Override
77 public void notifyDataSetChanged() {
78     notifyDataSetChanged();
79 }
80
81 @Override
82 public void onBindViewHolder(GlossaryItemViewHolder holder, int position) {
83     holder.bind(position);
84 }
85
86 @Override
87 public void onViewAttachedToWindow(GlossaryItemViewHolder holder) {
88     holder.onViewAttachedToWindow();
89 }
90
91 @Override
92 public void onViewDetachedFromWindow(GlossaryItemViewHolder holder) {
93     holder.onViewDetachedFromWindow();
94 }
95
96 @Override
97 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
98     recyclerView.setAdapter(this);
99 }
100
101 @Override
102 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
103 }
104
105 @Override
106 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
107 }
108
109 @Override
110 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
111     recyclerView.setAdapter(this);
112 }
113
114 @Override
115 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
116 }
117
118 @Override
119 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
120 }
121
122 @Override
123 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
124     recyclerView.setAdapter(this);
125 }
126
127 @Override
128 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
129 }
130
131 @Override
132 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
133 }
134
135 @Override
136 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
137     recyclerView.setAdapter(this);
138 }
139
140 @Override
141 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
142 }
143
144 @Override
145 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
146 }
147
148 @Override
149 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
150     recyclerView.setAdapter(this);
151 }
152
153 @Override
154 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
155 }
156
157 @Override
158 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
159 }
160
161 @Override
162 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
163     recyclerView.setAdapter(this);
164 }
165
166 @Override
167 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
168 }
169
170 @Override
171 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
172 }
173
174 @Override
175 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
176     recyclerView.setAdapter(this);
177 }
178
179 @Override
180 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
181 }
182
183 @Override
184 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
185 }
186
187 @Override
188 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
189     recyclerView.setAdapter(this);
190 }
191
192 @Override
193 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
194 }
195
196 @Override
197 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
198 }
199
200 @Override
201 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
202     recyclerView.setAdapter(this);
203 }
204
205 @Override
206 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
207 }
208
209 @Override
210 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
211 }
212
213 @Override
214 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
215     recyclerView.setAdapter(this);
216 }
217
218 @Override
219 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
220 }
221
222 @Override
223 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
224 }
225
226 @Override
227 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
228     recyclerView.setAdapter(this);
229 }
230
231 @Override
232 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
233 }
234
235 @Override
236 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
237 }
238
239 @Override
240 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
241     recyclerView.setAdapter(this);
242 }
243
244 @Override
245 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
245 }
246
247 @Override
248 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
249 }
250
251 @Override
252 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
253     recyclerView.setAdapter(this);
254 }
255
256 @Override
257 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
258 }
259
260 @Override
261 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
262 }
263
264 @Override
265 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
266     recyclerView.setAdapter(this);
267 }
268
269 @Override
270 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
271 }
272
273 @Override
274 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
275 }
276
277 @Override
278 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
279     recyclerView.setAdapter(this);
280 }
281
282 @Override
283 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
284 }
285
286 @Override
287 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
288 }
289
290 @Override
291 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
292     recyclerView.setAdapter(this);
293 }
294
295 @Override
296 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
296 }
297
298 @Override
299 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
300 }
301
302 @Override
303 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
304     recyclerView.setAdapter(this);
305 }
306
307 @Override
308 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
308 }
309
310 @Override
311 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
312 }
313
314 @Override
315 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
316     recyclerView.setAdapter(this);
317 }
318
319 @Override
320 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
320 }
321
322 @Override
323 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
324 }
325
326 @Override
327 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
328     recyclerView.setAdapter(this);
329 }
330
331 @Override
332 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
332 }
333
334 @Override
335 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
336 }
337
338 @Override
339 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
340     recyclerView.setAdapter(this);
341 }
342
343 @Override
344 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
344 }
345
346 @Override
347 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
348 }
349
350 @Override
351 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
352     recyclerView.setAdapter(this);
353 }
354
355 @Override
356 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
356 }
357
358 @Override
359 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
360 }
361
362 @Override
363 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
364     recyclerView.setAdapter(this);
365 }
366
367 @Override
368 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
368 }
369
370 @Override
371 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
372 }
373
374 @Override
375 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
376     recyclerView.setAdapter(this);
377 }
378
379 @Override
380 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
380 }
381
382 @Override
383 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
384 }
385
386 @Override
387 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
388     recyclerView.setAdapter(this);
389 }
390
391 @Override
392 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
392 }
393
394 @Override
395 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
396 }
397
398 @Override
399 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
400     recyclerView.setAdapter(this);
401 }
402
403 @Override
404 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
404 }
405
406 @Override
407 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
408 }
409
410 @Override
411 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
412     recyclerView.setAdapter(this);
413 }
414
415 @Override
416 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
416 }
417
418 @Override
419 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
419 }
420
421 @Override
422 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
423     recyclerView.setAdapter(this);
424 }
425
426 @Override
427 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
427 }
428
429 @Override
430 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
431 }
432
433 @Override
434 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
435     recyclerView.setAdapter(this);
436 }
437
438 @Override
439 public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
439 }
440
441 @Override
442 public void onLayout(RecyclerView recyclerView, LayoutParams layoutParams, int dx, int dy) {
443 }
444
445 @Override
446 public void onAttachedToRecyclerView(RecyclerView recyclerView) {
447     recyclerView.setAdapter(this);
448 }
```

Listing B.6: GlossaryAdapter.java

```

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

```

Listing B.7: GridEditorFragment.java

```

7 import android.support.v4.app.DialogFragment;
8 import android.support.v7.app.AlertDialog;
9 import android.text.Editable;
10 import android.text.TextWatcher;
11 import android.view.View;
12 import android.widget.EditText;

```

```

13 import android.widget.LinearLayout;
14 import de.muffinworks.knittingapp.R;
15 import de.muffinworks.knittingapp.util.Constants;
16
17 public class GridSizeDialogFragment extends DialogFragment {
18     private static final String BUNDLE_COLUMNS = "columns";
19     private static final String BUNDLE_ROWS = "rows";
20     private int mColumns = 0;
21     private int mRows = 0;
22     private OnGridSizeInteractionListener mListener;
23
24     public GridSizeDialogFragment() {}
25
26     public static GridSizeDialogFragment newInstance(int
27             columns, int rows) {
28         GridSizeDialogFragment fragment = new
29             GridSizeDialogFragment();
30         Bundle args = new Bundle();
31         args.putInt(BUNDLE_COLUMNS, columns);
32         args.putInt(BUNDLE_ROWS, rows);
33         fragment.setArguments(args);
34         return fragment;
35     }
36
37     @Override
38     public void onCreate(Bundle savedInstanceState) {
39         super.onCreate(savedInstanceState);
40         if (getArguments() != null) {
41             mColumns = getArguments().getInt(
42                     BUNDLE_COLUMNS);
43             mRows = getArguments().getInt(BUNDLE_ROWS);
44         }
45     }
46
47     @Override
48     public void onStart() {
49         super.onStart();
50         //http://stackoverflow.com/a/15619098/4738174
51         final AlertDialog dialog = (AlertDialog)
52             (dialog != null) ? dialog
53             : View.OnClickListener() {
54                 @Override
55                 public void onClick(View v) {
56                     int newColumns = Integer.parseInt(
57                         mColumnsEditText.getText().toString());
58                     if (newColumns != mColumns ||
59                         newRows != mRows) {
60                         onChartSizeSetResult(
61                             Integer.parseInt(
62                             mColumnsEditText.
63
64                     ) );
65                     }
66                 }
67             };
68
69             newColumns = Integer.parseInt(
70                 mColumnsEditText.getText().toString());
71             newRows = Integer.parseInt(
72                 mRowsEditText.getText().toString());
73             if (newColumns != mColumns || newRows != mRows) {
74                 mListener.onGridSizeChanged(newColumns, newRows);
75             }
76             mColumnsEditText.setText(String.valueOf(newColumns));
77             mRowsEditText.setText(String.valueOf(newRows));
78             mListener.onGridSizeChanged(newColumns, newRows);
79             mListener.onGridSizeChanged(newColumns, newRows);
80             mListener.onGridSizeChanged(newColumns, newRows);
81             mListener.onGridSizeChanged(newColumns, newRows);
82             mListener.onGridSizeChanged(newColumns, newRows);
83             mListener.onGridSizeChanged(newColumns, newRows);
84             mListener.onGridSizeChanged(newColumns, newRows);
85             mListener.onGridSizeChanged(newColumns, newRows);
86             mListener.onGridSizeChanged(newColumns, newRows);
87             mListener.onGridSizeChanged(newColumns, newRows);
88             mListener.onGridSizeChanged(newColumns, newRows);
89             mListener.onGridSizeChanged(newColumns, newRows);
90             mListener.onGridSizeChanged(newColumns, newRows);
91             mListener.onGridSizeChanged(newColumns, newRows);
92             mListener.onGridSizeChanged(newColumns, newRows);
93             mListener.onGridSizeChanged(newColumns, newRows);
94             mListener.onGridSizeChanged(newColumns, newRows);
95         }

```

```

96     }
97     .setNegativeButton(
98         R.string.dialog_cancel,
99         new DialogInterface.OnClickListener() {
100             @Override
101             public void onClick(DialogInterface dialog,
102                 int id) {
103                 // User cancelled the
104                 // dialog
105                 .setTitle(getResources().getString(R.
106                     string.dialog_title_grid_size));
107                 .setView(content);
108                 .create();
109                 return;
110             }
111             private void onChartSizeSetResult(int columns,
112                 int rows) {
113                 mListener.onSetCharSize(columns, rows);
114             }
115             @Override
116             public void onAttach(Context context) {
117                 super.onAttach(context);
118                 if (context instanceof
119                     OnGridSizeInteractionListener) {
120                     mListener = (OnGridSizeInteractionListener)
121                     context;
122                     throw new RuntimeException(context.toString()
123                         + " must implement "
124                         + "OnGridSizeInteractionListener");
125                 }
126             }
127             @Override
128             public void onDetach() {
129                 super.onDetach();
130                 mListener = null;
131             }
132             public interface OnGridSizeInteractionListener {
133                 void onSetCharSize(int columns, int rows);
134             }
135             class DimensionTextWatcher implements TextWatcher {
136                 private int oldValue;
137                 private EditText editText;
138                 private DimensionTextWatcher(EditText editText,
139                     int oldValue);
140             }
141         }
142         int oldValue) {
143             this.editText = editText;
144             this.oldValue = oldValue;
145         }
146         @Override
147         public void beforeTextChanged(CharSequence s,
148             int start, int count, int after) {}
149         @Override
150         public void onTextChanged(CharSequence s, int
151             start, int before, int count) {}
152         @Override
153         public void afterTextChanged(Editable s) {
154             if (s.length() == 0) {
155                 //no input
156                 ((AlertDialog)editDialog()).getButton(
157                     AlertDialog.BUTTON_POSITIVE).
158                     setEnabled(false);
159             } else if (Integer.parseInt(s.toString()) ==
160                 0) {
161                 //input is 0
162                 editText.setError(getString(R.string.
163                     error_dimension_zero));
164                 editText.setSelection(editText.length());
165                 ((AlertDialog)editDialog()).getButton(
166                     AlertDialog.BUTTON_POSITIVE).
167                     setEnabled(false);
168                 Constants.MAX_ROWS_AND_COLUMNS_LIMIT);
169             } else {
170                 ((AlertDialog)editDialog()).getButton(
171                     AlertDialog.BUTTON_POSITIVE).
172                     setEnabled(true);
173             }
174         }
175     }
176 }

```

Listing B.8: 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 de.muffinworks.knittingapp.util.Constants;
7
8 public abstract class KeyboardAdapterBase extends
9 BaseAdapter {
10
11     protected String[] mDescriptions;
12     protected String[][] mCharacters;
13     protected static LayoutInflater inflater = null;
14     protected Context mContext;
15     protected Snackbar mSnackbar = null;
16
17     public KeyboardAdapterBase(Context context) {
18
19         mDescriptions = Constants.SYMBOL_DESCRIPTIONS;
20         mCharacters = Constants.SYMBOLS;
21         mContext = context;
22
23     }

```

```

24     inflater = (LayoutInflater) context.
25     getSystemService(Context.LAYOUT_INFLATER_SERVICE);
26
27 }
28     @Override
29     public int getCount() {
30         return mCharacters.length;
31     }
32
33     @Override
34     public Object getItem(int position) {
35         return mCharacters[position];
36     }
37
38     @Override
39     public long getItemId(int position) {
40         return position;
41     }
42
43 }

```

Listing B.9: KeyboardAdapterBase.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.ViewGroup;
7 import de.muffinworks.knittingapp.R;
8 import de.muffinworks.knittingapp.Views;
9 import de.muffinworks.knittingapp.Views.KnittingFontButton;
10 public class KeyboardToggleAdapter extends
11     KeyboardAdapterBase {
12     public interface GridEditorKeyListener {
13         void onKeyToggled(String key);
14     }
15     private int mActiveKeyPosition = -1;
16     private GridEditorKeyListener mListener;
17
18     public KeyboardToggleAdapter(Context context,
19         GridEditorKeyListener listener) {
20         super(context);
21         mListener = listener;
22     }
23
24     public void setDeleteActive(boolean active) {
25         if (active) {
26             mActiveKeyPosition = -1;
27             notifyDataSetChanged();
28         }
29     }
30
31 }
32

```

```

33     @Override
34     public View getView(final int position, View
35         convertView, ViewGroup parent) {
36         KnittingFontButton key;
37         if (convertView == null) {
38             key = (KnittingFontButton) inflater.inflate(
39                 R.layout.view-grid-key, null);
40         } else {
41             key = (KnittingFontButton) convertView;
42         }
43         key.setActive(mActiveKeyPosition == position);
44         key.setText(mCharacters[position]);
45         key.setOnLongClickListener(new View.
46             OnLongClickListener() {
47                 @Override
48                 public boolean onLongClick(View v) {
49                     mSnackbar = Snackbar.make(v,
50                         mDescriptions[position], Snackbar.
51                         LENGTHLONG)
52                         .setAction(R.string.dialog_ok,
53                             new View.OnClickListener() {
54                             @Override
55                             public void onClick(View v) {
56                                 mSnackbar.dismiss();
57                             }
58                         });
59                     return true;
60                 }
61             });
62         key.setOnClickListener(new View.OnClickListener() {
63             @Override
64             public void onClick(View v) {
65                 mSnackbar.show();
66             }
67         });
68     }
69 }

```

Listing B.9: KeyboardAdapterBase.java

```
    } ;  
    @Override  
    public void onClick(View v) {  
        mActiveKeyPosition = Position;  
        mListener.onKeyToggled(mCharacters[  
            position]);  
        notifyDataSetChanged();  
    }  
}
```

Listing B.10: Key

Listing B.10: 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.widget.TextView;
7 import de.muffinworks.knittingapp.R;
8 import de.muffinworks.knittingapp.views.KnittingFontButton;
9
10 public class KeyboardTypingAdapter extends
11     KeyboardAdapterBase<String> {
12     public interface RowEditorKeyListener {
13         void onKeyClicked(String key);
14     }
15
16     private RowEditorKeyListener mListener;
17
18     public KeyboardTypingAdapter(Context context,
19         RowEditorKeyListener listener) {
20         super(context);
21         mListener = listener;
22     }
23
24     @Override
25     public View getView(final int position, View
26         convertView, ViewGroup parent) {
27
28         KnittingFontButton key;
29         if (convertView == null) {
30             key = (KnittingFontButton) inflater.inflate(
31                 R.layout.view_grid_key, null);
32             key = (KnittingFontButton) convertView;
33
34         key.setText(mCharacters[position]);
35         key.setOnLongClickListener(new View.
36             OnLongClickListener() {
37             @Override
38                 Snackbar onLongClick(View v) {
39                     mSnackbar = Snackbar.make(v,
40                         mDescriptions[position],
41                         LENGTHLONG)
42                         .setAction(R.string.dialog_ok,
43                             new View.OnClickListener() {
44
45                             @Override
46                             public void onClick(View v) {
47
48                             mSnackbar.show();
49                             return true;
45
46                         });
47
48                     key.setOnClickListener(new View.OnClickListener() {
49
50                         @Override
51                         public void onClick(View v) {
52                             mListener.onKeyClicked(mCharacters[5
53
54                         });
55
56                         return key;
57                     });
58                 }
59             );
60
61         }
62
63     }

```

Listing B.11: 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.TextView;  
7  
8 import de.muffinworks.knittingapp.R;  
9 import de.muffinworks.knittingapp.util.Constants;  
10
```

```
public class KnittingFontButton extends Button {  
    public KnittingFontButton(Context context) {  
        super(context);  
        init(context);  
    }  
    public KnittingFontButton(Context context,  
        AttributeSet attrs) {  
        super(context, attrs);  
        init(context);  
    }  
    private void init(Context context) {  
        // ...  
    }  
}
```

```

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

```

Listing B.12: KnittingFontButton.java

```

1 package de.muffinworks.knittingapp.views;
2 import android.content.Context;
3 import android.graphics.Point;
4 import android.graphics.Typeface;
5 import android.util.AttributeSet;
6 import android.widget.Layout;
7 import android.widget.AttributeSet;
8 import net.simplyadvanced.widgets.KeyboardlessEditText2;
9 import de.muffinworks.knittingapp.R;
10 import de.muffinworks.knittingapp.util.Constants;
11 public class LinedEditorEditText extends
12   KeyboardlessEditText2 {
13   public LinedEditorEditText(Context context,
14     AttributeSet attrs) {
15     super(context, attrs);
16     // set cursor visible and to beginning and
17     // request input focus
18     // needed to update the parents size
19     setCursorVisible(true);
20     requestFocus();
21     setSelection(0);
22     textSize = context.getResources();
23   }
24   private void getDimension(R dimen,
25     RowEditorDefaultTextSize);
26   public Point getCursorPosition() {
27     if (layout != null) {
28       public Point getCursorPosition() {
29         //https://stackoverflow.com/questions/5044342/how-to
30         //get-cursor-position-x-y-in-edittext-android
31         Layout layout = getLayout();
32         if (layout != null) {
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           return new Point(0, 0);
40         }
41     }
42   }

```

Listing B.12: KnittingFontButton.java

```

12 private int lines = 0;
13 public LineNumberTextView(Context context) {
14   super(context);
15   setTypeface(Typeface.createFromAsset(context,
16     getAssets(), Constants.KNITTINGFONT_PATH));
17   public LineNumberTextView(Context context,
18     AttributeSet attrs) {
19     super(context, attrs);
20   }

```

Listing B.13: LinedEditorEditText.java

```

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

```

Listing B.14: LineNumberTextView.java

```

23     private int measureLineNumbersTextWidth() {
24         return (int) getPaint().measureText(lines + "\n");
25     }
26
27     public int getExactWidth() {
28         return measureLineNumbersTextWidth() +
29             getPaddingRight() + getPaddingLeft();
30     }
31

```

```

23     public String getName() {
24         return name;
25     }
26
27     public void setName(String name) {
28         this.name = name;
29     }
30
31     public Metadata clone() {
32         Metadata m2 = new Metadata();
33         m2.id = id;
34         m2.name = name;
35         return m2;
36     }
37
38     @Override
39     public int compareTo(Metadata that) {
40         return this.name.compareTo(that.name);
41     }
42
43 }

```

Listing B.14: LineNumberTextView.java

```

1     package de.muffinworks.knittingapp.storage.models;
2     import java.util.UUID;
3
4     public class Metadata implements Comparable<Metadata> {
5         /**
6          * Used to identify the file this pattern is stored
7          * in.
8         */
9         private UUID id;
10        protected String name = "Default name";
11        public Metadata() {
12            id = UUID.randomUUID();
13        }
14
15        public String getFilename() {
16            return id + ".json";
17        }
18
19        public String getId() {
20            return id.toString();
21        }
22    }

```

```

18     super();
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.
28             parseRowToGridFormat(patternRows[0]).length;
29
30     }
31
32     public int getRows() {
33         return rows;
34     }

```

Listing B.15: Metadata.java

```

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

```

Listing B.16: Pattern.java

```

1    package de.muffinworks.knittingapp.fragments;
2    import android.app.Dialog;
3    import android.content.Context;
4    import android.content.DialogInterface;
5    import android.content.Intent;
6    import android.os.Bundle;
7    import android.support.annotation.NonNull;
8    import android.support.annotation.Nullable;
9    import android.support.annotation.RequiresApi;
10   import android.support.v4.app.DialogFragment;
11   import de.muffinworks.knittingapp.R;
12   public class PatternDeletedDialogFragment extends
13       DialogFragment {
14     private static final String BUNDLENAME = "name";
15     private OnPatternDeleteInteractionListener mListener;
16     private String mName = "";
17     private PatternDeletedDialogFragment() {}
18     public PatternDeletedDialogFragment() {}
19     public void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         new Instance(savedInstanceState) {
22             PatternDeletedDialogFragment fragment = new
23                 PatternDeletedDialogFragment();
24             Bundle args = new Bundle();
25             args.putString(BUNDLENAME, name);
26             fragment.setArguments(args);
27             fragment = fragment;
28     }
29     @Override
30     public void onCreate(@Nullable Bundle
31                         savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         if (getArguments() != null) {
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.dialog_title))
44             .setPositiveButton(R.string.dialog_delete,
45             new DialogInterface.OnClickListener()
46             {
47                 @Override
48                 public void onClick(DialogInterface
49                                     dialog, int which)
50                 {
51                     mListener.onConfirmDelete();
52                 }
53             })
54             .setNegativeButton(R.string.dialog_no,
55             new DialogInterface.OnClickListener()
56             {
57                 @Override
58                 public void onClick(DialogInterface
59                                     dialog, int which)
60                 {
61                     if (context instanceof
62                         OnPatternDeleteInteractionListener) {
63                         mListener =
64                             ((OnPatternDeleteInteractionListener)
65                             context);
66                     } else {
67                         throw new RuntimeException(context.toString());
68                     }
69                 }
70             })
71             .create();
72     }
73     @Override
74     public void onAttach(Context context) {
75         super.onAttach(context);
76         if (context instanceof
77             OnPatternDeleteInteractionListener) {
78             mListener =
79                 ((OnPatternDeleteInteractionListener)
80                 context);
81         }
82     }
83     @Override
84     public void onDetach() {
85         mListener = null;
86     }

```

Listing B.17: PatternDeleteDialogFragment.java

```

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

```

```

1  package de.muffinworks.knittingapp.views;
2
3  import android.content.Context;
4  import android.graphics.Canvas;
5  import android.graphics.Color;
6  import android.graphics.Paint;
7  import android.graphics.PointF;
8  import android.graphics.RectF;
9  import android.graphics.Typeface;
10 import android.util.AttributeSet;
11 import android.view.GestureDetector;
12 import android.view.MotionEvent;
13 import android.view.ScaleGestureDetector;
14 import android.view.View;
15 import de.muffinworks.knittingapp.R;
16 import de.muffinworks.knittingapp.util.Constants;
17 import de.muffinworks.knittingapp.util.PatternParser;
18 import de.muffinworks.knittingapp.util.PatternView;
19
20 public class PatternGridView extends View {
21
22     private static final String TAG = "GridEditorView";
23
24     private boolean canBeEdited = true;
25     private final float CELLWIDTH = 100.0f;
26     private final float MARGIN = 40.0f;
27     private final float ZOOMFACTORMIN = 0.5f;
28     private final float ZOOMFACTORMAX = 2.0f;
29     private final float DEFAULTSYMBOLTEXTSIZE = 60.0f;
30
31     private int rows = Constants.DEFAULTROWS;
32     private int columns = Constants.DEFAULTCOLUMNS;
33     private String[][] symbols = new String[columns][
34         rows];
35
36     /**
37      * represents the grid content
38      */
39     private RectF mContentRect = new RectF();
40
41     /**
42      * represents the visible area on the screen minus
43      * the padding
44      */
45     private RectF mCanvasRect = new RectF();
46     private Paint mRowHighlightPaint;
47     private Paint mGridPaint;

```

```

48     private Paint mLabelTextPaint;
49     private Paint mSymbolPaint;
50     private float mScaleFactor = 1f;
51     private ScaleGestureDetector mScaleGestureDetector;
52
53     private GestureDetector mGestureDetector;
54     private PointF mTranslationOffset = new PointF(0, 0);
55
56     private boolean hasScrolled = false;
57     private String mSelectedSymbol = null;
58     private int mCurrentRow = 0;
59
60     public PatternGridView(Context context) {
61
62         super(context);
63         init(context);
64     }
65
66     public PatternGridView(Context context, AttributeSet attrs) {
67         super(context, attrs);
68         init(context);
69     }
70
71     public PatternGridView(Context context, AttributeSet attrs, int defStyleAttr) {
72         super(context, attrs, defStyleAttr);
73         init(context);
74     }
75
76     private void init(Context context) {
77         initPaints();
78         updateContentRect();
79         mScaleGestureDetector = new ScaleGestureDetector(
80             context, new GridScaleListener());
81         mGestureDetector = new GestureDetector(context,
82             new GridGestureListener());
83
84         mContentRect.set(
85             MARGIN, MARGIN,
86             MARGIN + columns * CELL_WIDTH *
87             mScaleFactor,
88             MARGIN + rows * CELL_WIDTH *
89             mScaleFactor);

```

```

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

```

```

197     public void setCurrentRow (int newCurrentRow) {
198         mCurrentRow = newCurrentRow;
199         if (mCanvasRect.height () != 0.0 && mCurrentRow >
200             0) {
201             scrollCurrentRowToCenter ();
202         }
203     }
204     public void setCanBeEdited (boolean editable) {
205         canBeEdited = editable;
206     }
207     @Override
208     public boolean onTouchEvent (MotionEvent event) {
209         // see https://stackoverflow.com/questions-
210         // 9965695/how-to-distinguish-between-move-and-
211         // click-in-ontouchevent
212         if (event.getAction () == MotionEvent.ACTION_UP
213             && canBeEdited) {
214             if (mSelectedSymbol != null && !hasScrolled)
215                 float x = event.getX ();
216                 float y = event.getY ();
217                 int row = calculateRowFromValue (y);
218                 int column = calculateColumnFromValue (x)
219                 setSymbol (column, row);
220                 postInvalidate ();
221             } else {
222                 hasScrolled = false;
223             }
224         }
225     }
226     boolean retVal = mScaleGestureDetector.
227     onTouchEvent (event);
228     retVal = mGestureDetector.onTouchEvent (event) ||
229     retVal;
230     return retVal || super.onTouchEvent (event);
231 }
232 private int calculateRowFromValue (float y) {
233     return ((int) (y - mTranslationOffset.y - MARGIN
234             ) / (CELL_WIDTH * mScaleFactor));
235 }
236 private int calculateColumnFromValue (float x) {
237     return ((int) ((x - mTranslationOffset.x - MARGIN
238             ) / (CELL_WIDTH * mScaleFactor)));
239 }
240 private PointF getCellCenter (int column, int row) {
241     return new PointF (
242         MARGIN + column * CELL_WIDTH *
243             mScaleFactor +
244             CELL_WIDTH / 2 * mScaleFactor,
245             MARGIN + row * CELL_WIDTH *
246             mScaleFactor +
247             CELL_WIDTH / 2 * mScaleFactor +
248             CELL_WIDTH / 2 * mScaleFactor +
249             ((int) Math.abs (mSymbolPaint.
250             getFontMetrics ().top)) / 2
251     );
252 }
253     private float getPixelPositionTopForRow (int row) {
254         return mContentRect.top + (row - 1) * CELL_WIDTH *
255             mScaleFactor;
256 }
257     private float getPixelPositionBottomForRow (int row)
258         {
259             return mContentRect.top + row * CELL_WIDTH *
260                 mScaleFactor;
261         }
262     @Override
263     protected void onDraw (Canvas canvas) {
264         super.onDraw (canvas);
265         canvas.save ();
266         canvas.translate (mTranslationOffset.x,
267             mTranslationOffset.y);
268         if (mCurrentRow != 0) {
269             mContentRect.left =
270                 getPixelPositionTopForRow (
271                     mCurrentRow),
272                 mRowHighlightRect.left =
273                 getPixelPositionBottomForRow (
274                     mCurrentRow),
275                 drawGrid (canvas);
276                 drawAxisLabels (canvas);
277                 drawSymbols (canvas);
278             canvas.restore ();
279         }
280     }
281     private void drawSymbols (Canvas canvas) {
282         for (int c = 0; c < columns; c++) {
283             for (int r = 0; r < rows; r++)
284                 {
285                     String symbol = symbols [c] [r];
286                     if (symbol == null)
287                         mSymbolPaint.setTextSize (
288                             DEFAULT_SYMBOL_TEXTSIZE *
289                             mScaleFactor);
290                     PointF location = getCellCenter (c, r
291                         );
292                     canvas.drawText (
293                         symbol,
294                         location .x,
295                         location .y,
296                         mSymbolPaint
297                     );
298     }

```

```

296     }
297     }
298   }
299
300   private void drawGrid(Canvas canvas) {
301     if (rows > 0 && columns > 0) {
302       for (int i = 0; i < rows + 1; i++) {
303         canvas.drawLine(
304           (i * CELL_WIDTH * mScaleFactor)
305             + MARGIN,
306             (columns * CELL_WIDTH * mScaleFactor) + MARGIN,
307             (i * CELL_WIDTH * mScaleFactor) + MARGIN,
308             (i * CELL_WIDTH * mScaleFactor) + MARGIN,
309             mGridPaint);
310     }
311     for (int j = 0; j < columns + 1; j++) {
312       canvas.drawLine(
313         (j * CELL_WIDTH * mScaleFactor)
314           + MARGIN,
315             (j * CELL_WIDTH * mScaleFactor) + MARGIN,
316             (j * CELL_WIDTH * MARGIN,
317             (rows * CELL_WIDTH * mScaleFactor) + MARGIN,
318             mGridPaint);
319     }
320   }
321
322   private void drawAxisLabels(Canvas canvas) {
323     for (int r = 0; r < rows; r++) {
324       canvas.drawText(
325         //draw column labels
326         for (int text = r + 1;
327             text <= r + 1;
328             canvas.drawText(
329               text + "m",
330               r * CELL_WIDTH * mScaleFactor,
331               + CELL_WIDTH / 2 *
332                 mScaleFactor
333                 + ((int) Math.abs(
334                   mLabelTextPaint.
335                     getFontMetrics(
336                       "top") / 2
337                     + MARGIN));
338       ),
339     );
340   }
341
342   //draw row labels
343   for (int c = 0; c < columns; c++) {
344     int text = c + 1;
345     canvas.drawText(
346       text + "m",
347       c * CELL_WIDTH *
348         mScaleFactor
349       + MARGIN,
350         (columns * CELL_WIDTH * mScaleFactor)
351             + MARGIN,
352             (rows * CELL_WIDTH * mScaleFactor) + MARGIN,
353             (c * CELL_WIDTH * mScaleFactor) + MARGIN,
354             public void resetZoom() {
355               mScaleFactor = 1.0f;
356               invalidate();
357             }
358           class GestureDetector extends GestureDetector {
359             SimpleOnGestureListener {
360               @Override
361                 public boolean onScroll(MotionEvent e1,
362                               MotionEvent e2, float distanceX,
363                               float distanceY) {
364                 // minus operation because scroll is inverse
365                 // to dragging
366                 mTranslationOffset.x -= distanceX;
367                 mTranslationOffset.y -= distanceY;
368                 clampOffset();
369                 hasScrolled = true;
370                 postInvalidate();
371                 return true;
372               }
373             }
374             private void clampOffset() {
375               float maxRightOffset = mCanvasRect.width() -
376                 mContentRect.width() - 2 * MARGIN;
377               float maxDownOffset = mCanvasRect.height() - 2 * MARGIN;
378               if (mTranslationOffset.x > 0.0f ||
379                 maxRightOffset > 0) {
380                 mTranslationOffset.x = 0.0f;
381               }
382               if (mTranslationOffset.y > 0.0f ||
383                 maxDownOffset > 0) {
384                 mTranslationOffset.y = 0.0f;
385               }
386               if (maxRightOffset < 0 && mTranslationOffset.x <
387                 maxRightOffset) {
388                 mTranslationOffset.x = maxRightOffset;
389               }
390               if (maxDownOffset < 0 && mTranslationOffset.y <
391                 maxDownOffset) {
392                 mTranslationOffset.y = maxDownOffset;
393               }
394             }
395           }
396         }
397       }
398     }
399   }
400 }
```

```

392     }
393     class GridScaleListener extends ScaleGestureDetector
394     .SimpleOnScaleGestureListener {
395
396         private PointF viewportFocus = new PointF();
397         @Override
398         public boolean onScale(ScaleGestureDetector
399             detector) {
400
401             mScaleFactor *= detector.getScaleFactor();
402             mScaleFactor = Math.max(Math.min(
403                 mScaleFactor, ZOOMFACTOR_MAX),
404                 ZOOMFACTOR_MIN);
405
406             updateContentRect();
407             viewportFocus.set(
408                 detector.getFocusX(),
409                 detector.getFocusY());
410             invalidate();
411             return true;
412         }
413     }

```

Listing B.18: 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.Fragment;
6   import android.support.v4.app.FragmentManager;
7   import android.support.v4.widget.FloatingActionButton;
8   import android.support.v4.widget.ViewMenuHelper;
9   import android.view.Menu;
10  import android.view.MenuItem;
11  import android.view.View;
12  import android.widget.ListView;
13  import com.google.gson.JsonSyntaxException;
14  import java.io.IOException;
15  import java.util.List;
16  import javax.json.bind.Jsonb;
17  import de.muffinworks.knittingapp.fragments.*;
18  import de.muffinworks.knittingapp.storage.models.Pattern;
19  import de.muffinworks.knittingapp.util.Constants;
20  import de.muffinworks.knittingapp.views.adapters.*;
21  import de.muffinworks.knittingapp.views.adapters.PatternListAdapter;
22  public class PatternListActivity extends BaseActivity
23  implements PatternNameInteractionListener {
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          enableBackActionBar(false);
37
38          mPatternsList = (ListView) findViewById(R.id
39              patterns_list);

```

```

405         mAdapter = new PatternListAdapter(this);
406         mPatternsList.setAdapter(mAdapter);
407         mPatternsList.setItemsCanFocus(true);
408
409         mFab = (FloatingActionButton) findViewById(R.id.
410             fab);
411         mFab.setOnClickListener(new View.OnClickListener
412             () {
413                 @Override
414                 public void onClick(View v) {
415                     showSetNameDialog();
416                 }
417             });
418
419         requestExternalStoragePermission();
420
421     }
422
423     @Override
424     protected void onResume() {
425         super.onResume();
426         mAdapter.notifyDataSetChanged();
427         checkExportAvailability();
428     }
429
430     @Override
431     public void onCreateOptionsMenu(Menu menu) {
432         getMenuInflater().inflate(R.menu.
433             menu_pattern_list, menu);
434         mExportAllMenu = menu.findItem(R.id.export_all);
435         checkExportAvailability();
436     }
437
438     @Override
439     public void onOptionsMenuSelected(MenuItem item)
440

```

```

78     int id = item.getItemId();
79     if (id == R.id.importPattern) {
80         if (isExternalStoragePermissionGranted()) {
81             importFile();
82         } else {
83             requestExternalStoragePermission();
84         }
85     } else if (id == R.id.exportAll) {
86         if (isExternalStoragePermissionGranted()) {
87             exportAllPatterns();
88         } else {
89             requestExternalStoragePermission();
90         }
91     }
92     return super.onOptionsItemSelected(item);
93 }
94 private void exportAllPatterns() {
95     try {
96         mStorage.exportAll();
97         showAlertDialog(getString(R.string.success_export_all));
98     } catch (IOException e) {
99         showAlertDialog(getString(R.string.error_export));
100    }
101 }
102 private void importFile() {
103     Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
104     intent.setType("application/json");
105     startActivityForResult(intent, Constants.FILE_PICKER_REQUEST_CODE);
106 }
107 @Override
108 protected void onActivityResult(int requestCode, Intent resultCode, Intent data) {
109     if (requestCode == Constants.FILE_PICKER_REQUEST_CODE) {
110         if (data != null) {
111             try {
112                 final Pattern importedPattern =
113                     mStorage.loadFromFile(data.getData());
114                 if (mStorage.loadFromFile(data.getData()) == importedPattern) {
115                     showAlertDialog(getString(R.string.info_import_pattern_already_exists));
116                 }
117             } catch (Exception e) {
118                 Log.e("PatternListActivity", "Error importing pattern: " + e.getMessage());
119             }
120         }
121     }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }

import android.content.DialogInterface;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

```

Listing B.19: PatternListActivity.java

```

5 import android.content.DialogInterface;
6 import android.view.LayoutInflater;
7 import android.view.View;
8 import android.view.ViewGroup;
9 import android.widget.AdapterView;
10 import android.widget.ArrayAdapter;
11 import android.widget.ListView;
12 import android.widget.TextView;
13 import android.widget.Toast;
14 import android.widget.AdapterView.OnItemClickListener;
15 import android.widget.AdapterView.OnItemSelectedListener;
16 import android.widget.AdapterView.OnLongClickListener;
17 import android.widget.AdapterView.OnTouchListener;
18 import android.widget.AdapterView.OnItemSelectedListener;
19 import android.widget.AdapterView.OnLongClickListener;
20 import android.widget.AdapterView.OnTouchListener;
21 import android.widget.AdapterView.OnItemSelectedListener;
22 import android.widget.AdapterView.OnLongClickListener;
23 import android.widget.AdapterView.OnTouchListener;
24 import android.widget.AdapterView.OnItemSelectedListener;
25 import android.widget.AdapterView.OnLongClickListener;
26 import android.widget.AdapterView.OnTouchListener;
27 import android.widget.AdapterView.OnItemSelectedListener;
28 import android.widget.AdapterView.OnLongClickListener;
29 import android.widget.AdapterView.OnTouchListener;
30 import android.widget.AdapterView.OnItemSelectedListener;
31 import android.widget.AdapterView.OnLongClickListener;
32 import android.widget.AdapterView.OnTouchListener;
33 import android.widget.AdapterView.OnItemSelectedListener;
34 import android.widget.AdapterView.OnLongClickListener;
35 import android.widget.AdapterView.OnTouchListener;
36 import android.widget.AdapterView.OnItemSelectedListener;
37 import android.widget.AdapterView.OnLongClickListener;
38 import android.widget.AdapterView.OnTouchListener;
39 import android.widget.AdapterView.OnItemSelectedListener;
40 import android.widget.AdapterView.OnLongClickListener;
41 import android.widget.AdapterView.OnTouchListener;
42 import android.widget.AdapterView.OnItemSelectedListener;
43 import android.widget.AdapterView.OnLongClickListener;
44 import android.widget.AdapterView.OnTouchListener;
45 import android.widget.AdapterView.OnItemSelectedListener;
46 import android.widget.AdapterView.OnLongClickListener;
47 import android.widget.AdapterView.OnTouchListener;
48 import android.widget.AdapterView.OnItemSelectedListener;
49 import android.widget.AdapterView.OnLongClickListener;
50 import android.widget.AdapterView.OnTouchListener;
51 import android.widget.AdapterView.OnItemSelectedListener;
52 import android.widget.AdapterView.OnLongClickListener;
53 import android.widget.AdapterView.OnTouchListener;
54 import android.widget.AdapterView.OnItemSelectedListener;
55 import android.widget.AdapterView.OnLongClickListener;
56 import android.widget.AdapterView.OnTouchListener;
57 import android.widget.AdapterView.OnItemSelectedListener;
58 import android.widget.AdapterView.OnLongClickListener;
59 import android.widget.AdapterView.OnTouchListener;
60 import android.widget.AdapterView.OnItemSelectedListener;
61 import android.widget.AdapterView.OnLongClickListener;
62 import android.widget.AdapterView.OnTouchListener;
63 import android.widget.AdapterView.OnItemSelectedListener;
64 import android.widget.AdapterView.OnLongClickListener;
65 import android.widget.AdapterView.OnTouchListener;
66 import android.widget.AdapterView.OnItemSelectedListener;
67 import android.widget.AdapterView.OnLongClickListener;
68 import android.widget.AdapterView.OnTouchListener;
69 import android.widget.AdapterView.OnItemSelectedListener;
70 import android.widget.AdapterView.OnLongClickListener;
71 import android.widget.AdapterView.OnTouchListener;
72 import android.widget.AdapterView.OnItemSelectedListener;
73 import android.widget.AdapterView.OnLongClickListener;
74 import android.widget.AdapterView.OnTouchListener;
75 import android.widget.AdapterView.OnItemSelectedListener;
76 import android.widget.AdapterView.OnLongClickListener;
77 import android.widget.AdapterView.OnTouchListener;
78 import android.widget.AdapterView.OnItemSelectedListener;
79 import android.widget.AdapterView.OnLongClickListener;
80 import android.widget.AdapterView.OnTouchListener;
81 import android.widget.AdapterView.OnItemSelectedListener;
82 import android.widget.AdapterView.OnLongClickListener;
83 import android.widget.AdapterView.OnTouchListener;
84 import android.widget.AdapterView.OnItemSelectedListener;
85 import android.widget.AdapterView.OnLongClickListener;
86 import android.widget.AdapterView.OnTouchListener;
87 import android.widget.AdapterView.OnItemSelectedListener;
88 import android.widget.AdapterView.OnLongClickListener;
89 import android.widget.AdapterView.OnTouchListener;
90 import android.widget.AdapterView.OnItemSelectedListener;
91 import android.widget.AdapterView.OnLongClickListener;
92 import android.widget.AdapterView.OnTouchListener;
93 import android.widget.AdapterView.OnItemSelectedListener;
94 import android.widget.AdapterView.OnLongClickListener;
95 import android.widget.AdapterView.OnTouchListener;
96 import android.widget.AdapterView.OnItemSelectedListener;
97 import android.widget.AdapterView.OnLongClickListener;
98 import android.widget.AdapterView.OnTouchListener;
99 import android.widget.AdapterView.OnItemSelectedListener;
100 import android.widget.AdapterView.OnLongClickListener;
101 import android.widget.AdapterView.OnTouchListener;
102 import android.widget.AdapterView.OnItemSelectedListener;
103 import android.widget.AdapterView.OnLongClickListener;
104 import android.widget.AdapterView.OnTouchListener;
105 import android.widget.AdapterView.OnItemSelectedListener;
106 import android.widget.AdapterView.OnLongClickListener;
107 import android.widget.AdapterView.OnTouchListener;
108 import android.widget.AdapterView.OnItemSelectedListener;
109 import android.widget.AdapterView.OnLongClickListener;
110 import android.widget.AdapterView.OnTouchListener;
111 import android.widget.AdapterView.OnItemSelectedListener;
112 import android.widget.AdapterView.OnLongClickListener;
113 import android.widget.AdapterView.OnTouchListener;
114 import android.widget.AdapterView.OnItemSelectedListener;
115 import android.widget.AdapterView.OnLongClickListener;
116 import android.widget.AdapterView.OnTouchListener;
117 import android.widget.AdapterView.OnItemSelectedListener;
118 import android.widget.AdapterView.OnLongClickListener;
119 import android.widget.AdapterView.OnTouchListener;
120 import android.widget.AdapterView.OnItemSelectedListener;
121 import android.widget.AdapterView.OnLongClickListener;
122 import android.widget.AdapterView.OnTouchListener;
123 import android.widget.AdapterView.OnItemSelectedListener;
124 import android.widget.AdapterView.OnLongClickListener;
125 import android.widget.AdapterView.OnTouchListener;
126 import android.widget.AdapterView.OnItemSelectedListener;
127 import android.widget.AdapterView.OnLongClickListener;
128 import android.widget.AdapterView.OnTouchListener;
129 import android.widget.AdapterView.OnItemSelectedListener;
130 import android.widget.AdapterView.OnLongClickListener;
131 import android.widget.AdapterView.OnTouchListener;
132 import android.widget.AdapterView.OnItemSelectedListener;
133 import android.widget.AdapterView.OnLongClickListener;
134 import android.widget.AdapterView.OnTouchListener;
135 import android.widget.AdapterView.OnItemSelectedListener;
136 import android.widget.AdapterView.OnLongClickListener;
137 import android.widget.AdapterView.OnTouchListener;
138 import android.widget.AdapterView.OnItemSelectedListener;
139 import android.widget.AdapterView.OnLongClickListener;
140 import android.widget.AdapterView.OnTouchListener;
141 import android.widget.AdapterView.OnItemSelectedListener;
142 import android.widget.AdapterView.OnLongClickListener;
143 import android.widget.AdapterView.OnTouchListener;
144 import android.widget.AdapterView.OnItemSelectedListener;
145 import android.widget.AdapterView.OnLongClickListener;
146 import android.widget.AdapterView.OnTouchListener;
147 import android.widget.AdapterView.OnItemSelectedListener;
148 import android.widget.AdapterView.OnLongClickListener;
149 import android.widget.AdapterView.OnTouchListener;
150 import android.widget.AdapterView.OnItemSelectedListener;
151 import android.widget.AdapterView.OnLongClickListener;
152 }


```

```

import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;
import de.muffinworks.knittingapp.EditorActivity;
import de.muffinworks.knittingapp.R;
import de.muffinworks.knittingapp.ViewerActivity;
import de.muffinworks.knittingapp.PatternStorage;
import de.muffinworks.knittingapp.storage.Patterns;
import de.muffinworks.knittingapp.storage.Patterns;
import de.muffinworks.knittingapp.util.Constants;
public class PatternListAdapter extends BaseAdapter {
    private Context mContext;
    private Metadata[] mPatterns;
    private PatternStorage mStorage = PatternStorage.getInstance();
    private LayoutInflater mInflater;
    public PatternListAdapter(Context context) {
        mContext = context;
        mStorage.init(mContext);
        mPatterns = mStorage.listMetadataEntries();
        mInflater = (LayoutInflater) mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }
    @Override
    public int getCount() {
        return mPatterns.length;
    }
    @Override
    public Object getItem(int position) {
        return mPatterns[position];
    }
    @Override
    public long getItemId(int position) {
        return position;
    }
    @Override
    public void notifyDataSetChanged() {
        mPatterns = mStorage.listMetadataEntries();
        super.notifyDataSetChanged();
    }
    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {
        PatternItemViewHolder viewHolder;
        if (convertView == null) {
            convertView = mInflater.inflate(R.layout.
                view_item_list_pattern, null);
            viewHolder = new PatternItemViewHolder(
                convertView);
            convertView.setTag(viewHolder);
        } else {
            viewHolder = (PatternItemViewHolder) convertView.getTag();
        }
        viewHolder.itemView.set

```

106 storage.delete(id);
107 notifyDataSetChanged();
108 }
109 }
110 .setNegativeButton(R.string.dialog_no,
111 new DialogInterface.OnClickListener() {
112 @Override
113 public void onClick(DialogInterface dialog,
114 int which) {
115 dialog.cancel();
116 }
117 dialog.show();
118 }
119 }
120 static class PatternItemViewHolder {
121
122 public TextView mPatternName;
123 public ImageButton mEditButton;
124 public ImageButton mDeleteButton;
125 }
126 public PatternItemViewHolder(View root) {
127 mPatternName = (TextView) root.findViewById(
128 R.id.pattern_name);
129 mEditButton = (ImageButton) root.
130 findViewById(R.id.button_edit);
131 mDeleteButton = (ImageButton) root.
132 findViewById(R.id.button_delete);
133 }
134
135 }

39 }
40 @Override
41 public void onCreate(@Nullable Bundle
42 savedInstanceState) {
43 super.onCreate(savedInstanceState);
44 if (getArguments() != null) {
45 mName = getArguments().getString(BUNDLE_NAME
46);
47 }
48 }
49 }
50 @Override
51 public Dialog onCreateDialog(Bundle
52 savedInstanceState) {
53 final LinearLayout parent = (LinearLayout)
54 inflate(R.layout.view_pattern_name_input
55 , null);
56 final EditText input = (EditText) parent.
57 findViewById(R.id.input);
58 input.setText(mName);
59 input.setSelection(input.length());
60 input.setFilters(new InputFilter[] {
61 new InputFilter.LengthFilter(MAX_NAME_LENGTH) });
62 final AlertDialog dialog = new AlertDialog.
63 Builder(getActivity())
64 .setPositiveButton(R.string.dialog_ok,
65 new DialogInterface.OnClickListener() {
66 public void onClick(DialogInterface dialog,
67 int id) {
68 mListener.onSetName(input.
69 getText().toString());
70 }
71 })
72 .setNegativeButton(R.string.
73 negativeButton, new DialogInterface.
74 OnClickListener() {
75 public void onClick(DialogInterface dialog,
76 int id) {
77 mListener.onSetName(input.
78 getText().toString());
79 }
80 });
81 PatternNameDialogFragment.newInstance(
82 String name) {
83 PatternNameDialogFragment fragment = new
84 PatternNameDialogFragment();
85 Bundle args = new Bundle();
86 args.putString(BUNDLE_NAME, name);
87 fragment.setArguments(args);
88 }
89 return fragment;

Listing B.20: PatternListAdapter.java

```

dialog.cancel, new DialogInterface.  

    OnClickListener() {
66      public void onClick(DialogInterface dialog,  

67          int id) {
68        // User cancelled the dialog
69      }
70      .setTitle(getString(R.string.  

71          dialog_title_pattern_name))
72      .create();
73      input.addTextChangedListener(new TextWatcher() {
74        @Override
75        public void beforeTextChanged(CharSequence s  

76          , int start, int count, int after) {
77          @Override
78          public void onTextChanged(CharSequence s,  

79            int start, int before, int count) {}
80          @Override
81          public void afterTextChanged(Editable s) {
82            if (s.toString().isEmpty()) {
83              dialog.getButton(AlertDialog.BUTTON_POSITIVE).setEnabled(  

84                false);
85            } else {
86              dialog.getButton(AlertDialog.BUTTON_POSITIVE).setEnabled(true
87            );
88          }
89          dialog.setOnShowListener(new DialogInterface.  

90          OnShowListener() {
91            @Override
92            public void onShow(DialogInterface dialog) {
93              ((AlertDialog) dialog).getButton(  

94                  AlertDialog.BUTTON_POSITIVE).  

95                  setEnabled(false);
96            }
97          });
98          @Override
99          public void onAttach(Context context) {
100         if (context instanceof
101             OnPatternNameInteractionListener) {
102           mListener = (OnPatternNameInteractionListener)
103             OnPatternNameInteractionListener;
104           else {
105             throw new RuntimeException(context.toString()
106               + getString(R.string.  

107                 "OnPatternNameInteractionListener"));
108           }
109         }
110         @Override
111         public void onDetach() {
112           super.onDetach();
113           mListener = null;
114         }
115       }
116     }
117     public interface OnPatternNameInteractionListener {
118       void onSetName(String name);
119     }
120   }

```

Listing B.21: PatternNameDialogFragment.java

```

20      +"10"+Constants.EMPTY_SYMBOL
21      +"10"+Constants.EMPTY_SYMBOL
22      +"10"+Constants.EMPTY_SYMBOL
23      +"10"+Constants.EMPTY_SYMBOL;
24  private static final String REGEX_ALL_NUMBER_CHARACTER_PAIRS = "[0-9]* ([a-
25      oa-oz])";
26  private static final String REGEX_ALL_FORBIDDEN_CHARS = "-+-,!@#$%^&*()";
27  private static final String REGEX_LOOKBEHIND_LINEFEED = "(?<=\n)";
28  private static final String REGEX_ALL_DIGITS_END =
29  private static final String REGEX_ALL_DIGITS = "[\r\n]";
30

```

```

1  package de.muffinworks.knittingapp.util;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.regex.MatchResult;
5  import java.util.regex.Pattern;
6  import java.util.regex.PatternParser;
7
8  public class PatternParser {
9
10    private static final String EMPTY_STRING = "";
11    private static final String LINEFEED = "\n";
12    private static final String STRING =
13    DEFAULT_EMPTY_PATTERN_STRING =
14    +"10"+Constants.EMPTY_SYMBOL;
15    +"10"+Constants.EMPTY_SYMBOL;
16    +"10"+Constants.EMPTY_SYMBOL;
17    +"10"+Constants.EMPTY_SYMBOL;
18    +"10"+Constants.EMPTY_SYMBOL;
19    +"10"+Constants.EMPTY_SYMBOL;

```

APPENDIX B. SOURCE CODE

xxx

```

31     static private Pattern pattern = Pattern.compile(
32         REGEX_ALLNUMBERCHARACTERPAIRS);
33     public static String parseGridToRowFormat(String[][] input) {
34         if (input == null) return null;
35         String result = "";
36         for (int r = 0; r < input[0].length; r++) {
37             String previousSymbol = input[0][r]; // init
38             String currentSymbol = "; // first symbol in pattern
39             int count = 0;
40             for (int c = 0; c < input.length; c++) {
41                 currentSymbol = input[c][r];
42                 if (currentSymbol.equals(previousSymbol))
43                     count++;
44                 else {
45                     // append count and previousSymbol
46                     // save new previousSymbol
47                     // reset count
48                     result += count == 1 ? previousSymbol
49                     : previousSymbol + count +
50                     previousSymbol;
51                     previousSymbol = currentSymbol;
52                     count = 1;
53                 }
54                 result += count == 1 ? previousSymbol :
55                     count + previousSymbol;
56                 if (r != input[0].length - 1) {
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                 return result;
67             }
68             public static String[][] parseRowToGridFormat(String
69                 input) {
70                 if (input == null || input.isEmpty()) return
71                     null;
72                 // expanded row of 3h -> hh
73                 ArrayList<String> expandedRows = new ArrayList<
74                     >();
75                 // remove all characters that are not numeric or
76                 // used for the symbols font
77                 // https://stackoverflow.com/questions/1761051/
78                 // difference-between-n-and-r
79             }
80             int columns = 0;
81             for (int r = 0; r < compressedRows.length; r++)
82                 columns++;
83             if (columns == 0) return null;
84             ArrayList<String> groupedSymbols = new
85                 ArrayList<String>();
86             for (int i = 0; i < compressedRows.size(); i++)
87                 groupedSymbols.add(compressedRows.get(i));
88             String row = compressedRows.get(i).replaceAll(
89                 "\n", EMPTY);
90             row = row.replace(""\n", " ");
91             if (row.equals("") || row.equals(Constants.EMPTY_SYMBOL))
92                 groupedSymbols.add(group);
93             else {
94                 Matcher m = pattern.matcher(row);
95                 while(m.find()) {
96                     String group = m.group(0); // group 0 is
97                     // always entire match
98                     groupedSymbols.add(group);
99                 }
100            }
101            if (groupedSymbols.size() == 1) return groupedSymbols.get(0);
102            int symbolFactor = groupedSymbols.size();
103            int symbolCount = 0;
104            String expandedRow = "";
105            for (String group : groupedSymbols) {
106                String symbolFactorString = group.replaceAll(
107                    REGEX_ALLNONDIGITS, EMPTY);
108                if (symbolFactorString.equals("0"))
109                    symbol = group.replaceAll(
110                        REGEX_ALLDIGITS, EMPTY);
111                if (!symbolFactorString.isEmpty())
112                    expandedRow += symbol;
113                symbolFactor = Integer.parseInt(
114                    symbolFactor);
115            }
116            if ((symbolCount + factor) >
117                Constants.MAXROWSANDCOLUMNSLIMIT) {
118                symbolCount += factor;
119                for (int j = 0; j < factor; j++) {
120                    expandedRow += symbol;
121                }
122            }
123        }
124    }
125}
```

```

124     // only one symbol, not grouped e.g.
125     symbolCount++;
126     expandedRow += symbol;
127   }
128   }if (symbolCount > columns) columns =
129     symbolCount;
130   expandedRows.add(expandedRow);
131 }
132 String[][] result = new String[columns][
133   compressedRows.length];
134   // iterate over String [][] and fill in from row
135   strings
136   for (int r = 0; r < compressedRows.length; r++)
137     for (int c = 0; c < expandedRows.get(r).length();
138       result[c][r] = Character.toString(
139         expandedRows.get(r).charAt(c));
140       }
141       // fill null places with placeholder for empty
142       for (int c = 0; c < result.length; c++)
143         for (int r = 0; r < result[c].length; r++)
144           if (symbol == null)
145             result[c][r] = Constants.
146               EMPTY_SYMBOL;
147       String[] strings = result[c];
148       }
149     }
150   }
151   return result;
152 }
153 public static String parsePojoToRowFormat(String []
154   patternRows) {
155   if (patternRows == null) patternRows.length ==
156     0) return DEFAULT_EMPTY_PATTERN_STRING;
157   String result = "";
158   for (String row : patternRows) {
159     result += row + "\n";
160   }
161   return result;
162 }
163 public static String [] parseRowFormatToPojo(String
164   rowInput) {
165   // split after linefeed \n
166   String [] rows = rowInput.split(
167     REGEX_LOOKBEHIND_LINEFEED);
168   int [] symbolsPerRows = new int[rows.length];
169   int columns = 1;
170   for (int r = 0; r < rows.length; r++)
171     ArrayList<> groupedSymbols = new
172       ArrayList<>();
173     rows[r] = rows[r].replaceAll(
174       REGEX_ALL_DIGITS_END, EMPTY);
175     rows[r] = rows[r].replaceAll(LINEFEED, EMPTY);
176   }
177   StringBuilder sb = new StringBuilder();
178   int columnCount = 0;
179   for (MatchResult group : groupedSymbols) {
180     String symbolFactor = group.group(1);
181     String symbol = group.group(2);
182     if (columnCount >= Constants.
183       MAX_ROWS_AND_COLUMNS_LIMIT)
184       break;
185     if (!symbolFactor.isEmpty())
186       factor = Integer.parseInt(
187         symbolFactor);
188     int factor = Integer.parseInt(
189       symbolFactor);
190     if (factor + columnCount > Constants.
191       MAX_ROWS_AND_COLUMNS_LIMIT)
192       factor = Constants.
193         MAX_ROWS_AND_COLUMNS_LIMIT -
194         columnCount;
195     sb.append(factor);
196     else {
197       columnCount += factor;
198       symbolsPerRows[r] += factor;
199     }
200   }
201   return rows;
202 }
203 rows[r] = sb.toString();
204 if (columnCount > columns) columns =
205   columnCount;
206   // Append trailing empty symbols to ensure the
207   // right amount of columns
208   for (int r = 0; r < rows.length - r++);
209     if (diff == columns - symbolsPerRows[r]);
210       rows[r] += diff + Constants.EMPTY_SYMBOL;
211     }
212   else if (diff == 1)
213     rows[r] += Constants.EMPTY_SYMBOL;
214   }
215   return rows;
216 }
217 public static String [] parseGridFormatToPojo(String
218   [] gridInput) {
219   String rowFormat = parseGridToRowFormat(
220

```

gridInput);
 return parseRowFormatToPojo(rowFormat);
}

public static String[][] parsePojoToGridFormat(
 String[] patternRows)**{**
 if (patternRows == **null**) || patternRows.length ==
 0)**{**
 String[][] emptyDefaultPattern = **new** String[**1**][**1**];
 emptyDefaultPattern[**0**] = Constants.**DEFAULT_COLUMNS**[Constants.**DEFAULT_ROWS**];
 for (**int** c = **0**; c < Constants.**DEFAULT_COLUMNS**; c++) {
 for (**int** r = **0**; r < Constants.**DEFAULT_ROWS**; r++) {
 emptyDefaultPattern[c][r] = Constants.**EMPTY_SYMBOL**;
 }
 }
 }
 return emptyDefaultPattern;
}

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

public void convertToGrid_02() {
 String in = "**02h**";
 String[] expected = {
 "**0**",
 "**2**",
 "**h**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

public void convertToGrid_02x2() {
 String in = "**2h\n2y**";
 String[] expected = {
 "**2**",
 "**h**",
 "**2**",
 "**y**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

public void convertToGrid_2x2() {
 String in = "**2h\n3y**";
 String[] expected = {
 "**2**",
 "**h**",
 "**3**",
 "**y**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

public void convertToGrid_2x2_x3() {
 String in = "**2h\n3y\n2**";
 String[] expected = {
 "**2**",
 "**h**",
 "**3**",
 "**y**",
 "**2**",
 "**h**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

public void convertToGrid_2x2_x2() {
 String in = "**2h**";
 String[] expected = {
 "**2**",
 "**h**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

public void convertToGrid_2x2_x2_x2() {
 String in = "**2\n2\n2\n2**";
 String[] expected = {
 "**2**",
 "**\n2**",
 "**2**",
 "**\n2**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

public void convertToGrid_2x2_x2_x2_x2() {
 String in = "**2\n2\n2\n2\n2\n2**";
 String[] expected = {
 "**2**",
 "**\n2**",
 "**2**",
 "**\n2**",
 "**2**",
 "**\n2**"
 };
 String[][] result = PatternParser.
 parseRowToGridFormat(in);
 assertTrue(Arrays.deepEquals(result, expected));
}

Listing B.22: PatternParser.java

```

79
80     @Test
81     public void convertToGrid_emptyString() {
82         String in = "";
83         String[] expected = null;
84         String[] result = PatternParser.
85             parseRowToGridFormat(in);
86         assertEquals(expected == result);
87     }
88     @Test
89     public void convertToGrid_null() {
90         String in = null;
91         String[] expected = null;
92         String[] result = PatternParser.
93             parseRowToGridFormat(in);
94         assertEquals(expected == result);
95     }
96     @Test
97     public void convertToGrid_2_1_1x4() {
98         String in = "2hgt\\n4d";
99         String[] expected = {
100             {"h", "d", "d", "d"}, 
101             {"h", "d", "d", "d"}, 
102             {"g", "d", "d", "d"}, 
103             {"t", "d", "d", "d"}};
104     }
105     String[] result = PatternParser.
106         parseRowToGridFormat(in);
107     assertEquals(result, expected);
108 }
109     @Test
110     public void convertToGrid_3x2() {
111         String in = "2h\\n2d\\n2g";
112         String[] expected = {
113             {"h", "d", "g"}, 
114             {"h", "d", "g"}, 
115             {"h", "d", "g"}};
116     }
117     String[] result = PatternParser.
118         parseRowToGridFormat(in);
119     assertEquals(result, expected);
120 }
121     @Test
122     public void convertToGrid_3x2_withForbiddenChars() {
123         String in = "2h\\n2d\\n-+-@#$%&*();|<>\\n"; 
124         String[] expected = {
125             {"h", "d", "g"}, 
126             {"h", "d", "g"}, 
127             {"h", "d", "g"}};
128     }
129 }
130     @Test
131     public void convertToGrid_2x3_test() {
132         String in = "hdg\\nhdg\\n";
133     }
134     String[][] expected = {
135         {"h", "h"}, 
136         {"d", "d"}, 
137         {"g", "g"}};
138     String[][] result = PatternParser.
139         parseRowToGridFormat(in);
140     assertEquals(result, expected);
141 }
142     @Test
143     public void convertToString_3x2() {
144         String[] in = {"h", "d", "g"}, 
145         {"h", "d", "g"}, 
146         {"h", "d", "g"};
147     }
148     String expected = "2h\\n2d\\n2g";
149     String result = PatternParser.
150         parseGridToRowFormat(in);
151     assertEquals(expected.equals(result));
152 }
153     @Test
154     public void convertToString_3x2WithEmptySpacesInTheEnd() {
155         String[] in = {"h", "d", "g"}, 
156         {"h", "d", "g"}, 
157         {"h", "d", "g"}, 
158     }
159     String expected = "2h\\n2d\\n2";
160     String result = PatternParser.
161         parseGridToRowFormat(in);
162     assertEquals(expected.equals(result));
163 }
164     @Test
165     public void convertToString_3x2WithEmptySpaces() {
166         String[] in = {"h", "d", "g"}, 
167         {"h", "d", "g"}, 
168         {"h", "d", "g"};
169     }
170     String expected = "2\\n2\\n2";
171     String result = PatternParser.
172         parseGridToRowFormat(in);
173     assertEquals(expected.equals(result));
174 }
175     @Test
176     public void convertToString_1x1() {
177         String[] in = {"h", "d", "g"}, 
178         {"h", "d", "g"}, 
179         {"h", "d", "g"};
180     }
181     String expected = "h";
182     String result = PatternParser.
183         parseGridToRowFormat(in);
184     assertEquals(expected.equals(result));
185 }
186     @Test
187     public void rowsWithSameLengthsToPojo() {
188         String in = "2n\\n2g\\n2h";
189     }

```

```

189     ArrayList<String> expected = new ArrayList<>();
243
244     String [][] result = PatternParser.
245         parsePocoToGridFormat(in);
246
247     assertEquals(Arrays.deepEquals(result, expected));
248
249     @Test
250     public void pocoToRow() {
251         String [] in = {
252             "3h",
253             "f2g",
254             "2.j",
255         };
256         String expected = "3h\nf2g\n2.j";
257         String result = PatternParser.
258             parsePocoToRowFormat(in);
259
260         assertEquals(result.length == expected.size());
261
262         @Test
263         public void rowToPocoWithTrailingNumber() {
263
264             String [] in = {"3r2"};
265             String [] expected = {"3r"};
266
267             String [] result = PatternParser.
268                 parseRowFormatToPoco(in);
269
270             @Test
271             public void rowToGridWithTrailingNumber() {
272                 String in = "3r2";
273                 String [] expected = {"3r"};
274
275             }
276
277             String [][] result = PatternParser.
278                 parseRowFormatToGridFormat(in);
279
280             assertEquals(Arrays.deepEquals(result, expected));
281
282             @Test
283             public void rowToGridEmptyFirstRow() {
284                 String in = "\n3h";
285                 String [] expected = {"3h"};
286
287             }
288
289             String [][] result = PatternParser.
290                 parseRowFormatToPoco(in);
291
292         }
293
294         @Test
295         public void rowToGridEmptyRowInMiddle() {
295
296             String in = "4f\n3h";
297             String [] expected = {"4f",
298             "3h",
299         };

```

```

300     "3h."
301     String[] result = PatternParser.
302         parseRowFormatToPojo(in);
303     assertTrue(Arrays.deepEquals(result, expected));
304 }
305 @Test
306 public void rowToGridEmptyRowAtEnd() {
307     String[] in = "4f\n3h\n";
308     String[] result = PatternParser.
309         parseRowFormatToPojo(in);
310     assertEquals("3h\n4h", result[0]);
311     assertEquals("3h", result[1]);
312     assertEquals("3h", result[2]);
313     assertEquals("3h", result[3]);
314 }
315     String[] result = PatternParser.
316         parseRowFormat(in);
317     assertTrue(Arrays.deepEquals(result, expected));
318 }
319 @Test
320 public void rowToPojoEmptyRowAtEnd() {
321     String[] in = "\n4f\n3h\n";
322     String[] result = PatternParser.
323         parseRowFormatToPojo(in);
324     assertEquals("4f",
325                 result[0]);
326     assertEquals("3h",
327                 result[1]);
328     String[] result = PatternParser.
329         parseRowFormatToPojo(in);
330     assertTrue(Arrays.deepEquals(result, expected));
331 }
332 @Test
333 public void rowToPojoEmptyRow() {
334     String[] in = "\n";
335     String[] result = PatternParser.
336         parseRowFormatToPojo(in);
337     String[] result = PatternParser.
338         parseRowFormatToPojo(in);
339     assertTrue(Arrays.deepEquals(result, expected));
340 }
341 @Test
342 public void rowToGridEmptyRow() {
343
344     String in = "\n";
345     String[] result = PatternParser.
346         parseRowFormatToPojo(in);
347     assertEquals("PatternParser", result[0]);
348     assertTrue(Arrays.deepEquals(result, expected));
349 }
350
351     String[] result = PatternParser.
352         parseRowToGridFormat(in);
353     assertEquals("rowToGridEmptyRowMiddle()", result[0]);
354     assertEquals("3h\n4h", result[1]);
355     assertEquals("3h", result[2]);
356     assertEquals("3h", result[3]);
357     assertEquals("3h", result[4]);
358     assertEquals("3h", result[5]);
359     assertEquals("3h", result[6]);
360     assertEquals("3h", result[7]);
361     assertEquals("3h", result[8]);
362     assertEquals("3h", result[9]);
363     assertEquals("3h", result[10]);
364     assertEquals("rowToPojoMoreThanMaxColumns()", result[11]);
365     assertEquals("rowToGridMoreThanMaxColumns()", result[12]);
366     assertEquals("rowToGridMoreThanMaxColumns()", result[13]);
367     assertEquals("rowToGridMoreThanMaxColumns()", result[14]);
368     assertEquals("rowToGridMoreThanMaxColumns()", result[15]);
369     assertEquals("rowToGridMoreThanMaxColumns()", result[16]);
370     assertEquals("rowToGridMoreThanMaxColumns()", result[17]);
371     assertEquals("rowToGridMoreThanMaxColumns()", result[18]);
372     assertEquals("rowToGridMoreThanMaxColumns()", result[19]);
373     assertEquals("rowToGridMoreThanMaxColumns()", result[20]);
374     assertEquals("rowToGridMoreThanMaxColumns()", result[21]);
375     assertEquals("rowToGridMoreThanMaxColumns()", result[22]);
376     assertEquals("rowToGridMoreThanMaxColumns()", result[23]);
377     assertEquals("rowToGridMoreThanMaxColumns()", result[24]);
378     assertEquals("rowToGridMoreThanMaxColumns()", result[25]);
379     assertEquals("rowToGridMoreThanMaxColumns()", result[26]);
380     assertEquals("rowToGridMoreThanMaxColumns()", result[27]);
381     assertEquals("rowToGridMoreThanMaxColumns()", result[28]);
382 }

```

Listing B.23: PatternParserTest.java

```

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.lang.reflect.Type;
18 import java.nio.channels.FileChannel;
19 import com.google.gson.reflect.TypeToken;

```

```

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 import de.muffinworks.knittingapp.storage.models.Metadatas;
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> mMetaDataTable;
33     private static PatternStorage storage = new
34         PatternStorage();
35     public static PatternStorage getInstance() {
36         if (storage != null) {
37             return storage;
38         } else {
39             return new PatternStorage();
40         }
41     }
42     private PatternStorage() {}
43     public void init(Context context) {
44         this.mContext = context.getApplicationContext();
45         loadMetadata();
46     }
47     private String getApplicationDir() {
48         return mContext.getFilesDir().getAbsolutePath();
49     }
50     private String getFilePathInApplicationDir(String
51         fileName) {
52         return getApplicationDir() + "/" + fileName;
53     }
54     private String getApplicationDir() {
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 : mMetaDataTable.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_load_metadata));
76         File patternFile = new File(
77             getFilePathInApplicationDir(id + ".json"));
78         File file = new File(Environment.getExternalStorageDirectory(),
79             getAbsolutePath());
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 inStream = new FileInputStream(
95             src);
96         FileOutputStream outStream = new FileOutputStream(dst);
97         FileChannel inChannel = inStream.getChannel();
98         FileChannel outChannel = outStream.getChannel();
99         inChannel.transferTo(0, inChannel.size(), outChannel);
100        inStream.close();
101        outStream.close();
102    }
103    private void loadMetadata() {
104        mMetaDataTable = new HashMap<>();
105        try {
106            File file = new File(getApplicationDir(),
107                Constants.METADATAFILENAME);
108            FileReader fileReader = new FileReader(file)
109                .// https://sites.google.com/site/gson/gson-
110                // user-guide#IO-Collections-Examples
111                Type metadataType = new TokenType();
112                List<Metadata> metadata = gson.fromJson(
113                    fileReader, metadataType);
114                for (Metadata e : metadata) {
115                    try {
116                        fileReader.close();
117                    } catch (IOException e) {
118                        logError(mContext.getString(R.string.
119                            error_load_metadata));
120                    }
121                }
122            }
123        } catch (FileNotFoundException e) {
124            e.printStackTrace();
125        }
126        if (metadata != null) {
127            for (Metadata m : metadata) {
128                mMetaDataTable.put(m.getId(), m);
129            }
130        }
131    }
132 }

```

```

121
122     }
123     } catch (FileNotFoundException e) {
124         mMetadataTable = new HashMap<*>();
125         return;
126     }
127 }
128
129 private void updateMetadata() {
130     try {
131         File file = new File(getApplicationDir(),
132             Constants.METADATAFILENAME);
133         String json = mGson.toJson(mMetadataTable,
134             values());
135         FileWriter fileWriter = null;
136         fileWriter = new FileWriter(file);
137         fileWriter.write(json);
138         fileWriter.close();
139         e.setError("FileWriter error (" + R.string.
140             error_update_metadata));
141     }
142     public Metadata[] listMetadataEntries() {
143         Metadata[] m = mMetadataTable.values().toArray(
144             new Metadata[mMetadataTable.size()]);
145         if (m.length > 0) {
146             Arrays.sort(m);
147         }
148         return m;
149     }
150
151 public void save(Pattern pattern) {
152     try {
153         FileWriter fileWriter = new FileWriter(
154             getFilePathInApplicationDir(pattern.
155             getFilename()));
156         fileWriter.write(mGson.toJson(pattern));
157         fileWriter.close();
158         // call clone to put only metadata
159         // information into hashmap, not actual
160         // pattern related
161         // information -> needs less resources
162         mMetadataTable.put(pattern.getId(), pattern.
163             clone());
164         updateMetadata();
165     } catch (IOException e) {
166         Log.e(TAG, "error_save-pattern");
167         e.printStackTrace();
168     }
169     Pattern patternFromFile = loadFromFile(path);
170     if (patternFromFile != null) {
171         logError(mContext.getString(R.string.
172             error_file_not_found, path));
173         return null;
174     }
175     public boolean checkPatternDuplicate(Pattern pattern
176         ) {
177         String id = pattern.getId();
178         return mMetadataTable.containsKey(id);
179     }
180     public Pattern load(String id) throws
181         JsonSyntaxException {
182         loadFromFile(getFilePathInApplicationDir(
183             id + ".json"));
184     }
185     public void clearAll() {
186         for (Metadata m : mMetadataTable.values()) {
187             getFileFromApplicationDir(m.getFilename())
188                 .delete();
189         }
190     }
191     public void delete(Pattern pattern) {
192         getFileFromApplicationDir(pattern.getId());
193         mMetadataTable.remove(pattern.getId());
194     }
195     public void delete(Metadata pattern) {
196         delete();
197     }
198     public void delete(String id) {
199         delete();
200     }
201     public void delete(Metadata pattern) {
202         updateMetadata();
203     }
204     private File getFileFromApplicationDir(String
205         filename) {
206         return new File(getApplicationContext(), filename);
207     }
208     private void logError(String message) {
209         Log.e(TAG, message);
210     }
211 }
212
213 public Pattern loadFromFile(String path) throws

```

Listing B.24: PatternStorage.java

```

1  package de.muffinworks.knittingapp.storage;
2
3  import android.content.Context;
4  import android.os.Environment;
5  import android.support.test.InstrumentationRegistry;
6  import android.support.test.runner.AndroidJUnit4;
7
8  import org.junit.After;
9  import org.junit.Before;
10 import org.junit.Test;
11 import org.junit.runner.RunWith;
12
13 import java.io.File;
14 import java.io.IOException;
15
16 import de.muffinworks.knittingapp.storage.models.
17     Metadata;
18
19 import static org.junit.Assert.assertEquals;
20 import static org.junit.Assert.assertNotNull;
21 import static org.junit.Assert.assertTrue;
22 import static org.junit.Assert.fail;
23
24 @RunWith(AndroidJUnit4.class)
25 public class PatternStorageTest {
26
27     private Context context = InstrumentationRegistry.
28         getInstrumentation();
29     private TargetContext targetContext = getApplicationContext();
30
31     @Before
32     public void setup() throws IOException {
33         storage = PatternStorage.getInstance();
34         storage.init(context);
35         storage.clearAll();
36
37     }
38
39     @Test
40     public void saveAndLoadPatternTest() throws
41         IOException {
42         Pattern pattern = new Pattern();
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
55         storage.save(pattern);
56         File test = new File(context.getFilesDir(),
57             getPath().toString() + "/"
58             + pattern.getFilename());
59
60         Pattern p2 = storage.load(pattern.getId());
61         assertEquals(pattern.equals(p2), true);
62
63     }
64
65     @Test
66     public void listEntriesTest() throws IOException {
67         assertTrue(storage.listMetadataEntries().length
68             == 0);
69         for (int i = 1; i <= 5; i++) {
70             Pattern pattern = new Pattern();
71             storage.save(pattern);
72             assertTrue(storage.listMetadataEntries().length
73                 == 1);
74
75         }
76         saveAndLoadPatternTest();
77         PatternStorage storage2 = PatternStorage.
78             getInstance();
79         assertTrue(storage2.listMetadataEntries().length
80             > 0);
81
82         String id = storage2.listMetadataEntries()[0].
83             getId();
84         storage.delete(id);
85         for (Metadata m : storage2.listMetadataEntries()) {
86             if (m.getId() == storage2.listMetadataEntries()
87                 [fail()]);
88
89         }
90         @Test
91         public void exportTest() throws IOException {
92             Pattern pattern = new Pattern();
93             pattern.setName("scarf");
94             pattern.setCurrentRow(2);
95             pattern.setPatternRows(new String[] {
96                 "3e";
97                 "2er";
98                 "ttt";
99                 "3f"
100            });
101            storage.save(pattern);
102            File file = storage.export(pattern.getId());
103            assertTrue(file.exists());
104
105        }
106
107        @Test
108        public void importTest() throws IOException {
109            Pattern pattern = new Pattern();
110            pattern.setName("scarf");
111            pattern.setCurrentRow(2);
112            pattern.setPatternRows(new String[] {
113                "3e";
114                "2er",
115            });
116
117    }

```

113 "ttt",
114 "3f",
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
125 @Test
126 public void exportAllTest() throws IOException {
127 storage.exportAll();
128 for (Metadata md : storage.listMetadataEntries()
129 for (Metadata md : storage.listMetadataEntries()
130 File file = new File(Environment.getExternalStorage().
131 getExternalStoragePublicDirectory(
132 Environment.DIRECTORY_DOCUMENTS), md.getId() +
133 ".json");
134 assertTrue(file.exists());
135
136 @After
137 public void cleanUp() throws IOException {
138 storage.clearAll();
139 }
140 }
141 }

34 package de.muffinworks.knittingapp.fragments;
35
36 import android.os.Bundle;
37 import android.support.annotation.Nullable;
38 import android.support.design.widget.Snackbar;
39 import android.support.v4.app.Fragment;
40 import android.view.LayoutInflater;
41 import android.view.View;
42 import android.view.ViewGroup;
43 import android.widget.GridView;
44 import java.util.Arrays;
45
46 import de.muffinworks.knittingapp.R;
47 import de.muffinworks.knittingapp.PatternStorage;
48
49 import de.muffinworks.knittingapp.storage.models.Pattern;
50
51 public class RowEditorFragment extends Fragment
52 implements KeyboardTypingAdapter,
53 RowEditorKeyListener {
54
55 private static final String TAG = "RowEditorFragment";
56
57 public RowEditorLinearLayout mRowEditorView;
58 private Pattern mPattern;
59 private static RowEditorFragment getInstance(String
60 patternId) {
61 RowEditorFragment fragment = new
62 RowEditorFragment();
63 if (patternId != null) {
64 Bundle bundle = new Bundle();
65 bundle.putString("id", patternId);
66 fragment.setArguments(bundle);
67 }
68 return fragment;
69
70 @Override
71 public void onCreate(@Nullable Bundle savedInstanceState)
72 super.onCreate(savedInstanceState);
73 mRowEditorView = (RowEditorLinearLayout) findViewById(R.id.row_editor_container);
74 if (mPattern != null) {
75 mRowEditorView.setAdapter(mPattern.
76 getPatternRows());
77 mRowEditorView.setPattern(mPattern);
78 }
79
80 if (mPattern != null) {
81 GridView mKeyboard = (GridView) view.
82 findViewById(R.id.keyboard_gridview);
83 mKeyboard.setAdapter(new KeyboardTypingAdapter(
84 getActivity(), this));
85 }

Listing B.25: PatternStorageTest.java

```

91     */
92     public void onDelete() {
93         mRowEditorView.onDeletePressed();
94     }
95     /**
96      * emulates the enter key on the system's soft
97      * keyboard
98     */
99     public void onEnter() {
100        mRowEditorView.onEnterPressed();
101    }
102    /**
103     * int start = mRowEditorView.getText().getSelectionStart();
104     * getEditText().getSelectionStart();
105     * insert(start, number);
106    }
107    /**
108     * mPattern = mStorage.load(mPattern.getId());
109     * mRowEditorView.setPattern(mPattern);
110     * getPatternRows();
111    }
112    /**
113     * int start = mRowEditorView.getText().getSelectionStart();
114     * getEditText().getSelectionStart();
115     * mRowEditorView.getText().getEditText().insert(
116         start, key.toUpperCase());
117    }
118    */

119    /**
120     * emulates the backspace key on the system's soft
121     * keyboard
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190

```

Listing B.26: RowEditorFragment.java

```

22 import de.muffinworks.knittingapp.R;
23 import de.muffinworks.knittingapp.util.Constants;
24 import de.muffinworks.knittingapp.util.PatternParser;
25 import de.muffinworks.knittingapp.views.LineNumberTextView;
26 import de.muffinworks.knittingapp.views.LinedEditorEditText;
27 public class RowEditorLinearLayout extends LinearLayout
28 {
29     private static final String TAG = "RowEditorLinearLayout";
30     private LineNumberTextView editText;
31     private LinedEditorEditText editText;
32     private LineNumberTextView lineNumbers;
33     private boolean mIsBeingDragged = false;
34     private Point mLastScrollTo = new Point();
35     private Scroller mScroller;
36     private PointF mLastMotion = new PointF();
37     private VelocityTracker mVelocityTracker;
38
39

```

```

40     private int mTouchSlop;
41     private int mMinimumVelocity;
42     public RowEditorLinearLayout(Context context) {
43         super(context);
44         init(context);
45     }
46     public RowEditorLinearLayout(Context context,
47         AttributeSet attrs) {
48         super(context, attrs);
49     }
50     private void init(Context context) {
51         super(AttributeSet attrs, defStyleAttr) {
52             super(context, attrs, defStyleAttr);
53             init(context);
54         }
55     }
56     private void init(Context context) {
57         setOrientation(HORIZONTAL);
58         LayoutInflater inflater = LayoutInflater.from(
59             context);
60         inflater.inflate(R.layout.view_row_editor, this,
61             true);
62         lineNumbers = (LineNumberTextView) findViewById(
63             R.id.row_editor_line_numbers);
64         editText = (LinedEditorEditText) findViewById(R.
65             id.row_editor_edit_text);
66         editText.addTextChangedListener(new TextWatcher
67             () {
68             @Override
69             public void beforeTextChanged(CharSequence s,
70                 int start, int count, int after) {}
71         });
72         scrollToTextChange();
73     }
74     public void afterTextChange(Editable s) {
75     }
76     mScroller = new Scroller(context);
77     setFocusable(true);
78     setDescendantFocusability(
79         FOCUS_AFTER_DESCENDANTS);
80     setWillNotDraw(false);
81     final ViewConfiguration config =
82         ViewConfiguration.get(context);
83     mTouchSlop = config.getScaledTouchSlop();
84     mMinimumVelocity = config.
85     getScaledMinimumFlingVelocity();
86     //??
87     // editText.requestFocus();
88     editText.setSelection(editText.getText().length());
89     @Override
90     protected void onMeasure(int widthMeasureSpec, int
91         heightMeasureSpec) {
92         super.onMeasure(widthMeasureSpec,
93             heightMeasureSpec);
94         updateEditorLines();
95     }
96     public void updateEditorLines() {
97         mScroller.forceFinished(true);
98         int lineCount = editText.getLineCount();
99         lineNumbers.updateLineNumbers(lineCount);
100        editText.setMinWidth(getWidth() - lineNumbers.
101        getWidth());
102    }
103    public String[] getPattern() {
104        String patternString = editText.getText().
105        toString();
106        return PatternParser.parseRowFormatToPojo(
107            patternString);
108    }
109    public void setPattern(String[] patternRows) {
110        final String pattern = PatternParser.
111        parsePojoToRowFormat(patternRows);
112        // not updating textView when calling setText(
113        // post(new Runnable() {
114        //     @Override
115        //     public void run() {
116        //         editText.setText(pattern);
117        //     }
118        // });
119        public EditText getEditText() {
120            return editText;
121        }
122        public void disableEditable() {
123            editText.setCursorVisible(false);
124            editText.setFocusableInTouchMode(false);
125            editText.setSelectable(false);
126            editText.clearFocus();
127        }
128    }
129    public void onEnterPressed() {
130        if (editText.getLineCount() + 1 > Constants.
131        MAX_ROWS_AND_COLUMNS_LIMIT) {
132            Snackbar.make(this, getResources().getString(
133                R.string.info_max_rows, Constants.
134                MAX_ROWS_AND_COLUMNS_LIMIT), Snackbar.
135                LENGTH_SHORT).show();
136        } else {
137            editText.dispatchKeyEvent(new KeyEvent(
138                KeyEvent.ACTION_DOWN, KeyEvent.
139                KEYCODE_ENTER));
140        }
141    }
142 }

```

```

136     updateEditorLines();
137     scrollToTextChange();
138 }
139
140 public void onDeletePressed() {
141     editText.dispatchKeyEvent(new KeyEvent(KeyEvent.ACTION_DOWN, KeyEvent.KEYCODE_DEL));
142     updateEditorLines();
143 }
144
145     @Override
146     public boolean onTouchEvent(MotionEvent event) {
147         if (!canScroll()) return false;
148         if (mVelocityTracker == null) {
149             mVelocityTracker = VelocityTracker.obtain();
200             mVelocityTracker.addMovement(event);
201             final int action = event.getAction();
202             final float x = event.getX();
203             final float y = event.getY();
204             switch (action) {
205                 case MotionEvent.ACTION_DOWN:
206                     if (!mScroller.isFinished())
207                         mScroller.abortAnimation();
208                     break;
209                 case MotionEvent.ACTION_MOVE:
210                     int deltaX = (int) (mLastMotion.x - x);
211                     int deltaY = (int) (mLastMotion.y - y);
212                     mLastMotion.set(x, y);
213                     if (deltaX < 0) {
214                         if (getScrollX() < 0)
215                             deltaX = 0;
216                     }
217                     else if (deltaX > 0) {
218                         final int rightEdge = getWidth() -
219                             getPaddingRight();
220                         final int availableToScroll =
221                             getChildAt(1).getRight() -
222                             getChildAt(0).getRight();
223                         if (availableToScroll > 0) {
224                             deltaX = Math.min(
225                                 availableToScroll, deltaX);
226                         } else {
227                             deltaX = 0;
228                         }
229                     }
230                     final int bottomEdge = getHeight() -
231                         getPaddingBottom();
232                     final int availableToScroll =
233                         getChildAt(0).getBottom() -
234                             getChildAt(0).getBottomEdge();
235
236                     if (availableToScroll > 0) {
237                         if (deltaY < 0)
238                             deltaY = 0;
239                     }
240                     else if (deltaY > 0) {
241                         final int bottomEdge = getHeight() -
242                             getPaddingBottom();
243                         final int availableToScroll =
244                             getChildAt(0).getBottom() -
245                             getChildAt(0).getBottomEdge();
246
247                     }
248
249                     if (deltaY != 0 || deltaX != 0)
250                         scrollBy(deltaX, deltaY);
251
252                     break;
253
254                 case MotionEvent.ACTION_UP:
255                     final VelocityTracker velocityTracker =
256                         mVelocityTracker;
257                     velocityTracker.computeCurrentVelocity(
258                         1000);
259                     int initialXVelocity = (int)
260                         velocityTracker.getXVelocity();
261                     int initialYVelocity = (int)
262                         velocityTracker.getYVelocity();
263                     if ((Math.abs(initialXVelocity) +
264                         Math.abs(initialYVelocity)) >
265                             mMinimumVelocity) && getChildCount() > 0) {
266                         fling(-initialXVelocity, -
267                             initialYVelocity);
268                     }
269                     if (mVelocityTracker != null) {
270                         mVelocityTracker.recycle();
271                         mVelocityTracker = null;
272                     }
273                     break;
274
275                 case MotionEvent.ACTION_CANCEL:
276                     if (mVelocityTracker != null) {
277                         mVelocityTracker.recycle();
278                         mVelocityTracker = null;
279                     }
280                     return true;
281             }
282
283             private boolean canScroll() {
284                 int childCount = getChildCount();
285                 if (childCount > 0) {
286                     int childrenHeight = 0;
287                     for (int i = 0; i < childCount; i++) {
288                         View child = getChildAt(i);
289                         childrenHeight += child.getHeight();
290                         childrenWidth += child.getWidth();
291                     }
292                     return (getHeight() < childrenHeight +
293                         getPaddingTop() + getPaddingBottom() ||
294                         (getWidth() < childrenWidth +
295                             getPaddingRight() + getPaddingLeft()));
296                 }
297                 return false;
298             }
299
300         }
301     }
302
303     @Override
304     public boolean onInterceptTouchEvent(MotionEvent ev) {
305         final int action = ev.getAction();
306         if (action == MotionEvent.ACTION_MOVE) &&
307             scrollBy(deltaX, deltaY);
308     }
309
310 }

```

```

235     mIsBeingDragged) {
236         return true;
237     }
238     if (!canScroll())
239         mIsBeingDragged = false;
240     return false;
241 }
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 - mLastMotion.x);
247         final int yDiff = (int) Math.abs(y - mLastMotion.y);
248         if (xDiff > mTouchSlop || yDiff > mTouchSlop)
249             mIsBeingDragged = true;
250         break;
251     case MotionEvent.ACTION_DOWN:
252         mLastMotion.x = x;
253         mLastMotion.y = y;
254         mIsBeingDragged = !mScroller.isFinished()
255         () ;
256         break;
257     case MotionEvent.ACTION_UP:
258         // release drag
259         mIsBeingDragged = false;
260         break;
261         // only intercept motion events if we are
262         // dragging
263         return mIsBeingDragged;
264     @Override
265     protected int computeVerticalScrollRange() {
266         return getChildCount() == 0 ? getHeight() :
267             getChildDimensions().bottom;
268     }
269     @Override
270     protected int computeHorizontalScrollRange() {
271         return getChildCount() == 0 ? getWidth() :
272             getChildDimensions().right;
273     }
274     @Override
275     protected void measureChild(View child, int
276         parentWidthMeasureSpec, int
277         parentHeightMeasureSpec) {
278         ViewGroup.LayoutParams lp = child.
279             getLayoutParams();
280         int childWidthMeasureSpec;
281         childWidthMeasureSpec = getChildMeasureSpec(
282             parentWidthMeasureSpec, getPaddingLeft(),
283             childHeightMeasureSpec + getPaddingRight(),
284             lp.width);
285         makeMeasureSpec(0, MeasureSpec.UNSPECIFIED);
286         child.measure(childWidthMeasureSpec,
287             parentHeightMeasureSpec);
288     }
289     @Override
290     protected void measureChildWithMargins(View child,
291         int widthUsed,
292         int heightUsed
293         ) {
294         MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();
295         lp.childLayoutParams = child.getLayoutParams();
296         lp.height = child.getMeasuredHeight();
297         lp.width = child.getMeasuredWidth();
298         lp.leftMargin = child.getMarginLeft();
299         lp.topMargin = child.getMarginTop();
300         lp.rightMargin = child.getMarginRight();
301         lp.bottomMargin = child.getMarginBottom();
302         lp.layoutX = mScroller.getCurrX();
303         lp.layoutY = mScroller.getCurrY();
304         if (getChildCount() > 0) {
305             Rect childrenDimens =
306                 getChildDimensions();
307             scrollTo(clamp(x, getWidth() -
308                 getPaddingRight() - getPaddingLeft() -
309                 childrenDimens.width),
310                 clamp(y, getHeight() -
311                     getPaddingBottom() - getPaddingTop() -
312                     childrenDimens.height()));
313         } else {
314             scrollTo(x, y);
315         }
316     }
317     @Override
318     protected void onScrollChanged(int scrollX, int
319         scrollY) {
320         if (childCount > 0) {
321             int width = 0;
322             int height = getChildAt(0).getHeight();
323         }
324     }
325     private Rect getChildDimensions() {
326         int childCount = getChildCount();
327         if (childCount > 0) {
328             int width = 0;
329             int height = getChildAt(0).getHeight();
330         }
331     }
332     // Keep on drawing until the animation has
333     // finished.
334     postInvalidate();
335 }
336 }
```

```

324     for (int i = 0; i < childCount; i++) {
325         View child = getChildAt(i);
326         width += child.getWidth();
327     }
328     return new Rect(0, 0, width, height);
329 }
330 return null;
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 claim
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() - getPaddingRight() -
345             getPaddingLeft();
346         int right = getChildAt(1).getRight();
347         mScroller.fling(getScrollX(), getScrollY(),
348             velocityX, velocityY, 0, right - width,
349             0, bottom - height);
350     }
351 }
352 @Override
353 public void scrollTo(int x, int y) {
354     // we rely on the fact View.scrollTo calls
355     Rect childrenDimensions = getChildrenDimensions()
356     ();
357     x = clamp(x, getWidth() - getPaddingRight(),
358             - getPaddingLeft(), childrenDimensions.width
359             ());
360     y = clamp(y, getHeight() - getPaddingBottom(),
361             () - getPaddingTop(), childrenDimensions.
362             height());
363     if (x != getScrollX() || y != getScrollY())
364     {
365         mLastScrollTo.set(x, y);
366         super.scrollTo(x, y);
367     }
368 }
369
370     if (viewDimens + currentPos > childDimens) {
371         return childDimens - viewDimens;
372     }
373     return currentPos;
374 }
375 /**
376     * scroll behavior so far: checking if point is in
377     * visible area works and scrolls to point if it
378     * is not visible. edge behavior is faulty:
379     * bottom: scrollTo() will clamp if line gets added
380     * at the bottom, will only scroll to second to
381     * last line,
382     * since textView height has not been adjusted yet
383     * at that point
384     * right: {#LinedEditorEditText#getCursorPosition}
385     * gives wrong x position if character is added
386     * on last word from {@string/lorem-long-with-breaks}. I believe that is
387     * because a normal editText is made to wrap at the end of its width. Since I
388     * am supressing that behavior, and setting
389     * the editText's minimum width new after each
390     * interaction by calling {#updateEditorLines}, it
391     * is likely that the implementation gets confused. The
392     * x position that is returned is always
393     * where the added character would be if we wrapped
394     * the line.
395 */
396 public void scrollToTextChange() {
397     // add line numbers width to get total width
398     Point position = editText.getCursorPosition();
399     position.x = position.x + lineNumbers.getWidth();
400     Point center = getScreenCenter();
401     int scrollX = getScrollX();
402     int scrollY = getScrollY();
403     boolean visible = new Rect(scrollX, scrollY,
404         getWidth() + scrollX, getHeight() + scrollY)
405         .contains(
406             position.x,
407             position.y);
408     if (!visible) {
409         int x = position.x - center.x;
410         int y = position.y - center.y;
411         scrollTo(x, y);
412         invalidate();
413     }
414 }
415
416 private Point getScreenCenter(int width) {
417     return new Point(width / 2, getHeight() / 2);
418 }
419
420 }
421
422 Listing B.27: RowEditorLinearLayout.java

```

```

1 package de.muffinworks.knittingapp;
2 import android.app.Activity;
3 import android.os.Bundle;
4 import android.view.Menu;
5 import android.view.MenuItem;
6 import android.view.View;
7 import android.widget.FrameLayout;
8 import android.widget.ImageView;
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 java.io.IOException;
15 import java.io.IOException;
16 import java.io.IOException;
17 import java.util.List;
18 import de.muffinworks.knittingapp.layouts.*;
19 import de.muffinworks.knittingapp.models.Pattern;
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 PatternGridView mRowPattern;
26     private RowEditorLinearLayout mRowPattern;
27     private boolean mIsRowFormatActive = false;
28     private Pattern mPattern;
29     @Override
30     protected void onCreate(Bundle savedInstanceState) {
31         super.onCreate(savedInstanceState);
32         setContentView(R.layout.activity_viewer);
33     }
34     @Override
35     protected void onSaveInstanceState(Bundle savedInstanceState) {
36         enableActionBar(true);
37         String patternId = getIntent().getStringExtra(
38             Constants.EXTRA_PATTERN_ID);
39         if (patternId != null) {
40             mPattern = mStorage.load(patternId);
41             setActionBarTitle(mPattern.getName());
42             initEditors();
43             initCounter();
44         }
45     }
46     @Override
47     public boolean onOptionsItemSelected(MenuItem item)
48     {
49         int id = item.getItemId();
50         if (id == R.id.reset_row_counter) {
51             updateRowCounter(1);
52             mGridPattern.scrollCurrentRowToCenter();
53             return true;
54         } else if (id == R.id.switch_view_style) {
55             switchEditors();
56             return true;
57         } else if (id == R.id.scroll_current_row_to_center) {
58             mGridPattern.scrollCurrentRowToCenter();
59     } else if (id == R.id.open_editor) {
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     } else if (id == R.id.open_glossary) {
67         Intent intent = new Intent(this,
68             GlossaryActivity.class);
69         startActivity(intent);
70     } else if (id == R.id.export_pattern) {
71         return super.onOptionsItemSelected(item);
72     }
73     private void exportPattern() {
74         try {
75             mStorage.export(mPattern.getId());
76             showAlertDialog(getString(R.string.
77                 success_export_pattern), Constants.
78                 EXPORT_DIALOG);
79         } catch (IOException e) {
80             showAlertDialog(getString(R.string.
81                 error_export));
82         }
83     }
84     @Override
85     protected void onCreateOptionsMenu(Menu menu) {
86         getMenuInflater().inflate(R.menu.menu_viewer,
87             menu);
88     }
89     @Override
90     protected void onActivityResult(int requestCode, int
91         resultCode, Intent data) {
92         if (resultCode == Activity.RESULT_OK) {
93             mRowPattern.setPattern(mPattern.
94             getId());
95             mGridPattern.setPattern(mPattern.
96             getId());
97             setActionBarTitle(mPattern.getName());
98         } else if (resultCode == Activity.RESULT_CANCELED) {
99             if (data != null) {
100                 boolean wasPatternDeleted = data.
101                     getBoolean(Constants.
102                         EXTRA_PATTERN_DELETED,
103                         false);
104                 if (wasPatternDeleted) {
105
106
107
108
109
10
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
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
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
58
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
60
610
611
612
613
614
615
616
617
618
619
61
620
621
622
623
624
625
626
627
628
629
62
630
631
632
633
634
635
636
637
638
639
63
640
641
642
643
644
645
646
647
648
649
64
650
651
652
653
654
655
656
657
658
65
659
660
661
662
663
664
665
666
667
668
66
669
670
671
672
673
674
675
676
677
678
679
67
680
681
682
683
684
685
686
687
688
689
68
690
691
692
693
694
695
696
697
698
699
69
700
701
702
703
704
705
706
707
708
709
70
710
711
712
713
714
715
716
717
718
719
71
720
721
722
723
724
725
726
727
728
729
72
730
731
732
733
734
735
736
737
738
739
73
740
741
742
743
744
745
746
747
748
749
74
750
751
752
753
754
755
756
757
758
759
75
760
761
762
763
764
765
766
767
768
769
76
770
771
772
773
774
775
776
777
778
779
77
780
781
782
783
784
785
786
787
788
789
78
790
791
792
793
794
795
796
797
798
799
79
7
800
801
802
803
804
805
806
807
808
809
80
810
811
812
813
814
815
816
817
818
819
81
820
821
822
823
824
825
826
827
828
829
82
830
831
832
833
834
835
836
837
838
839
83
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
85
859
860
861
862
863
864
865
866
867
868
86
869
870
871
872
873
874
875
876
877
878
87
879
880
881
882
883
884
885
886
887
888
88
889
890
891
892
893
894
895
896
897
898
899
89
8
900
901
902
903
904
905
906
907
908
909
90
910
911
912
913
914
915
916
917
918
919
91
920
921
922
923
924
925
926
927
928
929
92
930
931
932
933
934
935
936
937
938
939
93
940
941
942
943
944
945
946
947
948
949
94
950
951
952
953
954
955
956
957
958
95
959
960
961
962
963
964
965
966
967
968
96
969
970
971
972
973
974
975
976
977
978
97
979
980
981
982
983
984
985
986
987
988
98
989
990
991
992
993
994
995
996
997
998
99
999
9
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
100
1010
1011
1012
1013
1014
1015
1016
1017
1018
101
1019
1020
1021
1022
1023
1024
1025
1026
1027
102
1028
1029
102
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
103
1040
1041
1042
1043
1044
1045
1046
1047
1048
104
1049
1050
1051
1052
1053
1054
1055
1056
1057
105
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
106
1068
1069
106
1070
1071
1072
1073
1074
1075
1076
1077
107
1078
1079
107
1080
1081
1082
1083
1084
1085
1086
1087
108
1088
1089
108
1090
1091
1092
1093
1094
1095
1096
1097
109
1098
1099
109
1100
1101
1102
1103
1104
1105
1106
1107
110
1108
1109
110
1110
1111
1112
1113
1114
1115
1116
1117
111
1118
1119
111
1120
1121
1122
1123
1124
1125
1126
1127
112
1128
1129
112
1130
1131
1132
1133
1134
1135
1136
1137
113
1138
1139
113
1140
1141
1142
1143
1144
1145
1146
114
1147
1148
114
1149
114
1150
1151
1152
1153
1154
1155
1156
1157
115
1158
1159
115
1160
1161
1162
1163
1164
1165
1166
1167
116
1168
1169
116
1170
1171
1172
1173
1174
1175
1176
1177
117
1178
1179
117
1180
1181
1182
1183
1184
1185
1186
1187
118
1188
1189
118
1190
1191
1192
1193
1194
1195
1196
1197
119
1198
1199
119
1200
1201
1202
1203
1204
1205
1206
1207
120
1208
1209
120
1210
1211
1212
1213
1214
1215
1216
1217
121
1218
1219
121
1220
1221
1222
1223
1224
1225
1226
1227
122
1228
1229
122
1230
1231
1232
1233
1234
1235
1236
1237
123
1238
1239
123
1240
1241
1242
1243
1244
1245
1246
1247
124
1248
1249
124
1249
124
1250
1251
1252
1253
1254
1255
1256
1257
125
1258
1259
125
1259
125
1260
1261
1262
1263
1264
1265
1266
1267
126
1268
1269
126
1269
126
1270
1271
1272
1273
1274
1275
1276
1277
127
1278
1279
127
1279
127
1280
1281
1282
1283
1284
1285
1286
1287
128
1288
1289
128
1289
128
1290
1291
1292
1293
1294
1295
1296
1297
129
1298
1299
129
1300
1301
1302
1303
1304
1305
1306
1307
130
1308
1309
130
1309
130
1310
1311
1312
1313
1314
1315
1316
1317
131
1318
1319
131
1319
131
1320
1321
1322
1323
1324
1325
1326
1327
132
1328
1329
132
1329
132
1330
1331
1332
1333
1334
1335
1336
1337
133
1338
1339
133
1339
133
1340
1341
1342
1343
1344
1345
1346
1347
134
1348
1349
134
1349
134
1350
1351
1352
1353
1354
1355
1356
1357
135
1358
1359
135
1359
135
1360
1361
1362
1363
1364
1365
1366
1367
136
1368
1369
136
1369
136
1370
1371
1372
1373
1374
1375
1376
1377
137
1378
1379
137
1379
137
1380
1381
1382
1383
1384
1385
1386
1387
138
1388
1389
138
1389
138
1390
1391
1392
1393
1394
1395
1396
1397
139
1398
1399
139
1400
1401
1402
1403
1404
1405
1406
1407
140
1408
1409
140
1409
140
1410
1411
1412
1413
1414
1415
1416
1417
141
1418
1419
141
1419
141
1420
1421
1422
1423
1424
1425
1426
1427
142
1428
1429
142
1429
142
1430
1431
1432
1433
1434
1435
1436
1437
143
1438
1439
143
1439
143
1440
1441
1442
1443
1444
1445
1446
1447
144
1448
1449
144
1449
144
1450
1451
1452
1453
1454
1455
1456
1457
145
1458
1459
145
1459
145
1460
1461
1462
1463
1464
1465
1466
1467
146
1468
1469
146
1469
146
1470
1471
1472
1473
1474
1475
1476
1477
147
1478
1479
147
1479
147
1480
1481
1482
1483
1484
1485
1486
1487
148
1488
1489
148
1489
148
1490
1491
1492
1493
1494
1495
1496
1497
149
1498
1499
149
1500
1501
1502
1503
1504
1505
1506
1507
150
1508
1509
150
1509
150
1510
1511
1512
1513
1514
1515
1516
1517
151
1518
1519
151
1519
151
1520
1521
1522
1523
1524
1525
1526
1527
152
1528
1529
152
1529
152
1530
1531
1532
1533
1534
1535
1536
1537
153
1538
1539
153
1539
153
1540
1541
1542
1543
1544
1545
1546
1547
154
1548
1549
154
1549
154
1550
1551
1552
1553
1554
1555
1556
1557
155
1558
1559
155
1559
155
1560
1561
1562
1563
1564
1565
1566
1567
156
1568
1569
156
1569
156
1570
1571
1572
1573
1574
1575
1576
1577
157
1578
1579
157
1579
157
1580
1581
1582
1583
1584
1585
1586
1587
158
1588
1589
158
1589
158
1590
1591
1592
1593
1594
1595
1596
1597
159
1598
1599
159
1600
1601
1602
1603
1604
1605
1606
1607
160
1608
1609
160
1609
160
1610
1611
1612
1613
1614
1615
1616
1617
161
1618
1619
161
1619
161
1620
1621
1622
1623
1624
1625
1626
1627
162
1628
1629
162
1629
162
1630
1631
1632
1633
1634
1635
1636
1637
163
1638
1639
163
1639
163
1640
1641
1642
1643
1644
1645
1646
1647
164
1648
1649
164
1649
164
1650
1651
1652
1653
1654
1655
1656
1657
165
1658
1659
165
1659
165
1660
1661
1662
1663
1664
1665
1666
1667
166
1668
1669
166
1669
166
1670
1671
1672
1673
1674
1675
1676
1677
167
1678
1679
167
1679
167
1680
1681
1682
1683
1684
1685
1686
1687
168
1688
1689
168
1689
168
1690
1691
1692
1693
1694
1695
1696
1697
169
1698
1699
169
1700
1701
1702
1703
1704
1705
1706
1707
170
1708
1709
170
1709
170
1710
1711
1712
1713
1714
1715
1716
1717
171
1718
1719
171
1719
171
1720
1721
1722
1723
1724
1725
1726
1727
172
1728
1729
172
1729
172
1730
1731
1732
1733
1734
1735
1736
1737
173
1738
1739
173
1739
173
1740
1741
1742
1743
1744
1745
1746
1747
174
1748
1749
174
1749
174
1750
1751
1752
1753
1754
1755
1756
1757
175
1758
1759
175
1759
175
1760
1761
1762
1763
1764
1765
1766
1767
176
1768
1769
176
1769
176
1770
1771
1772
1773
1774
1775
1776
1777
177
1778
1779
177
1779
177
1780
1781
1782
1783
1784
1785
1786
1787
178
1788
1789
178
1789
178
1790
1791
1792
1793
1794
1795
1796
1797
179
1798
1799
179
1800
1801
1802
1803
1804
1805
1806
1807
180
1808
1809
180
1809
180
1810
1811
1812
1813
1814
1815
1816
1817
181
1818
1819
181
1819
181
1820
1821
1822
1823
1824
1825
1826
1827
182
1828
1829
182
1829
182
1830
1831
1832
1833
1834
1835
1836
1837
183
1838
1839
183
1839
183
1840
1841
1842
1843
1844
1845
1846
1847
184
1848
1849
184
1849
184
1850
1851
1852
1853
1854
1855
1856
1857
185
1858
1859
185
1859
185
1860
1861
1862
1863
1864
1865
1866
1867
186
1868
1869
186
1869
186
1870
1871
1872
1873
1874
1875
1876
1877
187
1878
1879
187
1879
187
1880
1881
1882
1883
1884
1885
1886
1887
188
1888
1889
188
1889
188
1890
1891
1892
1893
1894
1895
1896
1897
189
1898
1899
189
1900
1901
1902
1903
1904
1905
1906
1907
190
1908
1909
190
1909
190
1910
1911
1912
1913
1914
1915
1916
1917
191
1918
1919
191
1919
191
1920
1921
1922
1923
1924
1925
1926
1927
192
1928
1929
192
1929
192
1930
1931
1932
1933
1934
1935
1936
1937
193
1938
1939
193
1939
193
1940
1941
1942
1943
1944
1945
1946
1947
194
1948
1949
194
1949
194
1950
1951
1952
1953
1954
1955
1956
1957
195
1958
1959
195
1959
195
1960
1961
1962
1963
1964
1965
1966
1967
196
1968
1969
196
1969
196
1970
1971
1972
1973
1974
1975
1976
1977
197
1978
1979
197
1979
197
1980
1981
1982
1983
1984
1985
1986
1987
198
1988
1989
198
1989
198
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
199
1998
1999
199
2000
2001
2002
2003
2004
2005
2006
2007
200
2008
2009
200
2009
200
2010

```

```

102         finish() ;
103     }
104     }
105     }
106   }
107 }
108 private void initCounter () {
109     mRowText = (TextView) findViewById(R. id. row) ;
110     updateRowCounter();
111 }
112 ImageButton mIncreaseRow = (ImageButton)
113     findViewById(R. id. button_increase);
114     mIncreaseRow.setOnClickListener(new View.
115     OnClickListener() {
116         @Override
117         public void onClick(View v) {
118             updateRowCounter(mCurrentRow +1);
119         }
120     ImageButton mDecreaseRow = (ImageButton)
121     findViewById(R. id. button_decrease);
122     mDecreaseRow.setOnClickListener(new View.
123     OnClickListener() {
124         @Override
125         public void onClick(View v) {
126             updateRowCounter(mCurrentRow -1);
127         }
128     }
129     private void updateRowCounter(int rows) {
130         int maxRows = mPattern == null ? Constants.
131             DEFAULT_ROWS : mPattern.getRows();
132         mCurrentRow = Math. min(Math. max(rows, 1),
133             maxRows);
134         mRowText.setText(Integer. toString(mCurrentRow));
135         mPattern.setRowIndex(mCurrentRow);
136         mStorage. save(mPattern);
137         if (mGridPattern != null) {
138             mGridPattern.setCurrentRow(mCurrentRow);
139         }
140     }
141 }
142     private void updateRowCounter () {
143         if (mPattern != null) {
144             mCurrentRow = mPattern .getCurrentRow();
145             updateRowCounter (mCurrentRow);
146         }
147     }
148 }
149     private void initEditors () {
150         mPatternContainer = (FrameLayout) findViewById(R.
151             .id. editor_container);
152         mGridPattern = new PatternGridView(this);
153         mGridPattern.setCanBeEdited(false);
154         mGridPattern.setPattern(mPattern, getPatternRows(
155            ()));
156         mRowPattern = new RowEditorLinearLayout(this);
157         mRowPattern.disableEditable();
158     }
159 }
160     private void switchEditors () {
161         if (!mIsRowFormatActive) {
162             mPatternContainer.removeAllViews();
163             mPatternContainer.addView(mGridPattern);
164         } else {
165             mPatternContainer.removeAllViews();
166             mPatternContainer.addView(mRowPattern);
167             mRowPattern.setPatternRows();
168         }
169     }
170     private void removeAllViews () {
171         mPatternContainer.addView(mGridPattern);
172         mGridPattern.setPatternRows();
173     }
174 }
175 }
```

Listing B.28: ViewerActivity.java

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android:layout_width="match_parent"
        android:layout_height="match_parent">
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

```

8
9
10
11
12

Listing B.29: activity_editor.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:layout_width="match_parent"
6     android:layout_height="match_parent">
7
8   <ListView
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
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64

```

Listing B.30: activity-glossary_list.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2   <CoordinatorLayout
3     android:id="@+id/Coord"
4       xmlns:android="http://schemas.android.com/apk/res/
5         android:app="http://schemas.android.com/apk/res-auto"
6           xmlns:tools="http://schemas.android.com/tools"
7             android:layout_width="match_parent"
8               android:layout_height="match_parent"
9                 android:fitsSystemWindows="true"
10                   android:context="de.muffinworks.knittingapp.
11                     PatternListActivity"
12
13   <ListView
14     android:id="@+id/patterns_list"
15     android:layout_width="match_parent"
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
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64

```

Listing B.31: activity-pattern_list.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3   xmlns:android="http://schemas.android.com/apk/
4     res"
5     android:app="http://schemas.android.com/apk/res-auto"
6       android:orientation="vertical"
7         android:layout_width="match_parent"
8           android:layout_height="match_parent"
9             android:layout_weight="11"
10
11   <FrameLayout
12     android:id="@+id/editor_container"
13       android:textSize="80sp"
14         android:gravity="center"
15           android:layout_width="0dp"
16             android:layout_height="0dp"
17             android:layout_weight="8"/>
18
19   <FrameLayout
20     android:background="@color/colorPrimary"
21       android:layout_width="match_parent"
22         android:layout_height="3dp"/>
23
24   <LinearLayout
25     android:layout_width="match_parent"
26       android:layout_height="0dp"
27         android:orientation="horizontal"
28           android:id="@+id/containerRows"
29             android:layout_gravity="left|center_vertical"
30               android:layout_width="0dp"
31                 android:layout_height="match_parent"
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64

```

Listing B.32: activity-glossary.xml

```

65   <ImageButton
66     android:id="@+id/button_increase"
67     android:elevation="4dp"
68     android:layout_width="150dp"
69     android:layout_height="150dp"
70     android:background="@drawable/ripple_round"
71     android:src="@drawable/ic_add_white_24dp" />
72   </LinearLayout>
73 </LinearLayout>
74
75   <ImageButton
76     android:id="@+id/button_decrease"
77     android:elevation="4dp"
78     android:layout_marginLeft="75dp"
79     android:layout_width="75dp" />
80
81 </resources>
82
83 </LinearLayout>
84 </LinearLayout>
85 </LinearLayout>
86 </LinearLayout>
87 </LinearLayout>

```

Listing B.32: activity_viewer.xml

```

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

```

Listing B.33: attr.xml

```

18   <color name="primaryText">#212121</color>
19   <color name="secondaryText">#727272</color>
20   <color name="divider">#B6B6B6</color>
21   <color name="offWhite">#dedede</color>
22   <color name="black_01">#33dedede</color>
23   <color name="black_01">#33dedede</color>
24   <color name="black_20">#335757</color>
25   <color name="black_20">#335757</color>
26   <color name="black_30">#332929</color>
27   <color name="black_40">#665757</color>
28   <color name="black_60">#995575</color>
29   <color name="black_80">#cc5555</color>
30   <color name="offBlack">#2f2f2f</color>
31   <color name="red_400">#EF5350</color>
32
33 </resources>

```

Listing B.34: colors.xml

```

17 <?xml version="1.0" encoding="utf-8"?>
18 <LinearLayout
19   xmlns:android="http://schemas.android.com/apk/res/
20   android:orientation="horizontal"
21   android:orientation="vertical"
22   android:gravity="center"
23   android:weightSum="2"
24   android:layout_width="match_parent"
25   android:layout_height="match_parent">
26   <EditText
27     android:id="@+id/editText_columns"
28     android:dimeOptions="actionOnNext"
29     android:inputType="number"
30     android:digits="1234567890"
31     android:layout_width="100dp"
32     android:layout_height="50dp" />
33
34 </LinearLayout>

```

```

35   <EditText
36     android:id="@+id/edittext_rows"
37     android:imeOptions="actionDone"
38     android:inputType="number"
39     android:layout_height="1dp"
40     android:gravity="center"
41     android:layout_weight="1" />
42
43   <TextView
44     android:textSize="20sp"
45     android:layout_marginRight="15dp"
46     android:text="@string/rows"
47     android:gravity="right"
48     android:layout_width="wrap_content"
49     android:layout_height="wrap_content" />

```

Listing B.35: dialog_set_grid_size.xml

```

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

```

Listing B.36: dimens.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:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:weightSum="4" />
8
9   <de.murfinworks.knittingapp.views.PatternGridView
10    android:id="@+id/grid"
11    android:layout_width="match_parent"
12    android:layout_height="0dp"
13    android:layout_weight="3" />
14
15   <FrameLayout
16    android:layout_width="match_parent"
17    android:layout_height="3dp"
18    android:background="@color/colorPrimary" />
19
20   <LinearLayout
21    android:orientation="horizontal"
22    android:background="@color/colorPrimary" />
23
24   <Gridview
25    android:layout_width="match_parent"
26    android:layout_height="1dp"
27    android:weightSum="7" />
28
29   android:id="@+id/keyboard_gridview"
30   android:background="@color/colorAccent"

```

Listing B.38: fragment_editor_grid.xml

```

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

```

Listing B.37: dimens-w820.xml

```

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

```

Listing B.36: dimens.xml

APPENDIX B. SOURCE CODE

1

```

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       android:weightSum="3">
8
9   <de.muffinworks.knittingapp.layouts.
10    RowEditorLinearLayout
11      android:id="@+id/row_editor_container"
12        android:layout_weight="2"
13        android:layout_width="match_parent"
14        android:layout_height="0dp"/>
15
16   <FrameLayout
17     android:background="@color/colorPrimary"
18     android:layout_weight="match_parent"
19     android:layout_height="3dp"/>
20
21   <LinearLayout
22     android:layout_width="match_parent"
23     android:layout_height="7dp"
24     android:weightSum="1"
25
26     <GridView
27       android:id="@+id/keyboard_gridview"
28         android:background="@color/colorAccent"
29         android:scrollbarSize="10dp"
30         android:scrollbars="false"
31         android:horizontalSpacing="20dp"
32         android:layout_width="0dp"
33         android:layout_height="match_parent"
34         android:layout_weight="3"
35         android:numColumns="3"/>
36
37     <FrameLayout
38       android:background="@color/colorAccentDark"
39       android:layout_width="3dp"
40       android:layout_height="match_parent" />
41
42   <include
43     layout="@layout/view_numpad" />
44
45   <ImageButton
46     android:layout_width="0dp"
47     android:layout_height="0dp"
48     android:layout_weight="1"
49     android:background="@color/colorPrimary"
50     android:layout_width="0dp"
51     android:layout_height="match_parent"
52     android:layout_weightSum="2"
53     android:layout_weight="1" />
54
55   <ImageButton
56     android:layout_width="0dp"
57     android:layout_height="0dp"
58     android:layout_weight="1"
59     android:layout_width="match_parent"
60     android:src="@drawable/ic_backspace_white_24dp"
61     android:id="@+id/op_delete"
62     android:style="@style/FontButtonStyle"
63     android:contentDescription="Delete"
64     android:onClick="onDelete" />
65
66   <ImageButton
67     android:layout_width="0dp"
68     android:layout_height="0dp"
69     android:layout_weight="1"
70     android:layout_width="match_parent"
71     android:layout_weight="1"
72     android:src="@drawable/ic_keyboard_return_white_24dp"
73     android:id="@+id/op_enter"
74     android:style="@style/FontButtonStyle"
75     android:contentDescription="Enter"
76     android:onClick="onEnter" />
77
78   </LinearLayout>
79
80 </LinearLayout>
81 </LinearLayout>
82 </LinearLayout>
83 </fragment>
84
85 <?xml version="1.0" encoding="utf-8"?>
86 <menu
87   xmlns:android="http://schemas.android.com/apk/res/
88     android:orderInCategory="100"
89       android:title="@string/menu_export_pattern"
90       android:icon="@drawable/ic_swap_horiz_black_24dp"
91       android:showAsAction="never" />
92
93 <item
94   android:id="@+id/set_size"
95   android:orderInCategory="100"
96   android:savePattern="100" />
97
98 <item
99   android:id="@+id/save_pattern"
100  android:orderInCategory="100"
101    android:title="@string/menu_export_pattern"
102    android:icon="@drawable/ic_swap_horiz_black_24dp"
103    android:showAsAction="always" />
104
105 <item
106   android:id="@+id/switch_editor"
107   android:orderInCategory="100"
108     android:icon="@drawable/ic_menu_open_glossary"
109       android:title="@string/menu_open_glossary"
110       android:showAsAction="never" />
111
112 <item
113   android:id="@+id/switch_editor"
114   android:orderInCategory="100"
115     android:title="@string/menu_save_pattern"
116     android:showAsAction="always" />
```

Listing B.39: fragment_editor_row.xml

APPENDIX B. SOURCE CODE

```

35      android:orderInCategory="100"
36      android:icon="@drawable/ic_save_black_24dp"
37      android:title="@string/menu_save"
38      android:showAsAction="always" />
39
40      <item android:id="@+id/edit-pattern_name"
41          android:orderInCategory="100"
42          android:title="@string/menu_edit_name"
43          android:showAsAction="never" />
44
45

```

```

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
6     <item android:id="@+id/export_all"
7         android:orderInCategory="100"
8         android:title="@string/menu_export_all"
9         app:showAsAction="never" />
10

```

Listing B.40: menu_editor.xml

```

46      <item android:id="@+id/delete_pattern"
47          android:orderInCategory="100"
48          android:title="@string/menu_delete_pattern"
49          android:showAsAction="never" />
50
51

```

```

11     <item android:id="@+id/import_pattern"
12         android:orderInCategory="100"
13         android:title="@string/menu_import_pattern"
14         android:showAsAction="never" />
15
16

```

Listing B.41: menu_editor_pattern_list.xml

```

23     <item android:id="@+id/open_editor"
24         android:orderInCategory="100"
25         android:title="@string/menu_open_editor"
26         android:showAsAction="never" />
27
28     <item android:id="@+id/scroll_current_row_to_center"
29         android:orderInCategory="100"
30         android:title="@string/menu_open_editor"
31         android:showAsAction="always" />
32
33     <item android:id="@+id/switch_view_style"
34         android:orderInCategory="100"
35         android:title="@string/menu_jump_to_current_row"
36         android:showAsAction="always" />
37
38     <item android:id="@+id/icon_change"
39         android:orderInCategory="100"
40         android:title="@string/menu_switch_editor"
41         android:showAsAction="always" />
42

```

Listing B.42: menu_viewer.xml

```

9 <string name="dialog_title_grid_size">Change grid size</
10 <string name="dialog_title_pattern_delete">Delete
11 <string name="dialog_title_pattern_name">Pattern name</
12 <string name="dialog_title_pattern_save_changes">Save
13 <string name="dialog_title_pattern_save_changes" />
14
15 <resources>
16     <string name="app_name">Knitting App</string>
17     <!-- dialog related -->
18     <string name="dialog_ok">OK</string>
19     <string name="dialog_yes">Yes</string>
20     <string name="dialog_no">No</string>
21     <string name="dialog_cancel">Cancel</string>
22     <string name="dialog_cancel">Cancel</string>

```

```

14 <!--Activity titles-->
15 <string name="activity-title-glossary">Glossar</string>
16 <!--errors-->
17 <string name="error_over_max_size">Only %1$s supported</
18 string>
19 <string name="error_must_implement_interface">
20 <string translatable="false"> must implement %$s</string>
21 <string name="error_load_metadata">Could not close file
22 <string metadata="/string"> Could not write
23 <string pattern_to_disk</string>
24 <string name="error_file_not_found">Could not find file:
25 %1$s</string>
26 <string name="error_dimension_zero">Must be at least 1</
27 string>
28 <string name="error_external_storage_not_writable">
29 External storage is not writable</string>
30 <string name="error_external_storage_not_mounted">
31 External storage not available</string>
32 <string name="error_export">Export failed</string>
33 <string name="error_import_no_json">Import failed: can\'t
34 read file</string>
35 <!--success-->
36 <string name="success_export_pattern">Successfully
37 exported pattern to folder %1$s on SD card</string>
38 <string name="success_export_all">Successfully exported
39 to folder %1$s on SD card</string>
40 <string name="success_save_pattern">Saving successful</
41 string>
42 <string name="info_import_pattern_already_exists">This
43 pattern already exists. Do you want to overwrite the
44 existing file?</storage>
45 <string name="info_storage_permission">Allow access to
46 storage to export and import patterns</string>
47 <string name="info_max_rows">Only %1$d rows are
48 supported</string>
49 <!--resources-->
50 <string name="tag-dialog_fragment_edit_name">translateable
51 <string name="tag-dialog_fragment_delete_pattern">
52 <string name="tag-dialog_fragment_set_name">translatable
53 <string name="tag-dialog_fragment_set_name"/>
54 <string name="menu-grid-size">dialog_fragment_grid_size</
55 string>
56 <string name="menu-delete-pattern">Delete pattern</
57 string>
58 <string name="menu-reset-counter">Reset counter</string>
59 <string name="menu-open-glossary">Open glossary</string>
60 <string name="menu-open-editor">Edit pattern</string>
61 <string name="menu-jump-to-current-row">Current row</
62 string>
63 <string name="rows">rows</string>
64 <string name="columns">columns</string>
65 <string name="placeholder_pattern_name">Pattern name</
66 string>
67 <string name="placeholder_line_numbers">translatable =
68 false</string>
69 </resources>
70 <string name="error_export">Fehler beim exportieren</
71 string>
72 <string name="error_external_storage_not_mounted">
73 Externer Speicher nicht verfügbar</string>
74 <string name="error_external_storage_not_writable">
75 Externer Speicher kann nicht beschrieben werden!</
76 string>
77 <string name="error_file_not_found">Datei: %1$s konnte
78 nicht gefunden werden!</string>
79 <string name="error_load_metadata">File reader konnte
80 nicht geschlossen werden</string>
81 <string name="error_must_implement_interface">Nur %1$s unterstützt<
82 /string>
83 <string name="error_save_pattern">Muster konnte nicht
84 gespeichert werden</string>
85 <string name="error_update_metadata">Metadata konnte
86 nicht gespeichert werden</string>
87 <string name="info_import_string">Muster überschreiben?</
88 string>
89 <string name="error_over_max_size">Nur %1$d unterstützt<
90 /string>
91 <string name="dialog_title_löschen">Strickmuster
92 Strickmustername</string>
93 <string name="dialog_title_abbrechen">Abbrechen</string>
94 <string name="dialog_nein">Nein</string>
95 <string name="dialog_ok">OK</string>
96 <string name="dialog_title_größe ändern">Gittergröße ändern
97 <string name="dialog_title_pattern_delete">Strickmuster
98 <string name="dialog_wirklich_löschen">String>
99 <string name="dialog_title_pattern_name">
100 <string name="dialog_cancel">Abbrechen</string>
101 <string name="dialog_no">Nein</string>
102 <string name="dialog_ok">OK</string>
103 <string name="dialog_ja">Ja</string>
104 <string name="error_dimension_zero">Muss mindestens 1
105 sein</string>
106 <!--xml version="1.0" encoding="utf-8"?>
107 <resources>
108 <string name="app_name">Knitting App</string>
109 <string name="activity_title_glossary">Glossar</string>
110 <string name="columns">Spalten</string>
111 <string name="dialog_cancel">Abbrechen</string>
112 <string name="dialog_no">Nein</string>
113 <string name="dialog_speichern">Speichern?</string>
114 <string name="dialog_yes">Ja</string>
115 <string name="error_dimension_zero">Muss mindestens 1
116 <string name="info_over_max_size">Nur %1$d unterstützt<
117 <string name="error_save_pattern">Muster konnte nicht
118 gespeichert werden</string>
119 <string name="error_update_metadata">Metadata konnte
120 nicht gespeichert werden</string>
121 <string name="info_import_string">Muster überschreiben?</
122 string>
123 <string name="error_over_max_size">Nur %1$d unterstützt<
124 /string>

```

Listing B.43: strings.xml

```

25   <string name="info_max_rows">Nur maximal %1$d Reihen
26   <string name="info_storage_permission">Zugriff erlauben
27   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">MustergröÙe</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>

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

```

Listing B.44: strings-de.xml

```

37   <string name="menu_save">Speichern</string>
38   <string name="menu_switch_editor">Ansicht wechseln</string>
39   <string name="placeholder_line_numbers" translatable="false">
40     <string name="placeholder_pattern_name">Mustername</string>
41     <string name="rows">Reihen</string>
42     <string name="success_export_all">Muster erfolgreich
43       nach %1$s auf SD Karte exportiert</string>
44     <string name="success_save_pattern">Speichern
45       erfolgreich</string>
46     <string name="success_export_pattern">Muster erfolgreich
47       nach %1$s auf SD Karte exportiert</string>
48     <string name="error_import_no_json">Import
49       fehlgeschlagen: Datei kann nicht gelesen werden. </string>
50
51
52
53
54
55

```

Listing B.45: styles.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources xmlns:android="http://schemas.android.com/apk/res
3   <style name="AppTheme.NoActionBar">
4     <item name="windowActionBar">false</item>
5     <item name="windowNoTitle">true</item>
6     <item name="windowDrawsSystemBarBackgrounds"
7       >true</item>

```

Listing B.46: styles-port.xml

```

8       <style name="FontButtonStyle.KeyEvent">
9         <item name="android:layout_margin">8dp</item>
10        <item name="android:textSize">23sp</item>
11        <item name="android:fontStyle">
12          <resources>
13            <item name="statusBarColor">@android:color/
14              transparent</item>
15            </resources>
16          </style>
17        </resources>
18      </style>
19    </resources>
20  </resources>
21
22  <?xml version="1.0" encoding="utf-8"?>
23  <de.muffinworks.knittingapp.views.KnittingFontButton
24    xmlns:android="http://schemas.android.com/apk/res/
25      style="@style/FontButtonStyle.Key"
26      android:padding="30dp"

```

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

```

6       <item name="statusBarColor">@android:color/
7         transparent</item>
8       <item name="match_parent">
9         <de.muffinworks.knittingapp.views.KnittingFontButton>
10        <?xml version="1.0" encoding="utf-8"?>
11        <de.muffinworks.knittingapp.views.KnittingFontButton
12          xmlns:android="http://schemas.android.com/apk/res/
13            style="@style/FontButtonStyle.Key"
14            android:padding="30dp"

```

Listing B.48: view-grid_key.xml

```

14       <Textview
15         android:text="k"
16         android:gravity="center" />
17       <Textview
18         android:id="@+id/symbol_description"
19         android:layout_width="match_parent"
20         android:layout_height="match_parent"
21         android:padding="10dp"
22         android:gravity="left|center"
23         android:textSize="26sp" />
24       <LinearLayout
14       <?xml version="1.0" encoding="utf-8"?>
15       <LinearLayout
16         xmlns:android="http://schemas.android.com/apk/res/
17           android:orientation="horizontal"
18           android:layout_width="match_parent"
19           android:layout_height="match_parent">
20             <TextView
21               android:id="@+id/symbol"
22               android:layout_width="75dp"
23               android:layout_height="75dp"
24               android:layout_margin="16dp"
25               android:textSize="32sp"

```

Listing B.49: view_item_glossary.xml

```

12       <Imagebutton
13         android:id="@+id/pattern_name"
14         android:text="@string/placeholder_pattern_name"
15         android:layout_width="1"
16         android:layout_height="1"
17         android:layout_weight="1"
18         android:background="@color/colorAccent" />
19       <Imagebutton
20         android:background="@drawable/
21           keyboard_button_background"
22         android:id="@+id/button_edit" />

```

```

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     <TextView
8       android:gravity="center_vertical"
9       android:textSize="25sp"
10      android:layout_gravity="start | center_vertical"
11      android:layout_margin="32sp"

```

```

23   android:src="@drawable/ic_mode_edit_black_24dp"
24   android:layout_gravity="center_vertical"
25   android:layout_width="50dp"
26   android:layout_height="match_parent" />
27
28 <ImageButton
29   android:tint="@color/colorAccent"
30   android:background="@drawable/
31   keyboard_button_background" data-bbox="235 160 254 821"/>
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>
39

```

Listing B.50: view_item_list_pattern.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2   <com.android.calculator2.CalculatorPadLayout
3     xmlns:android="http://schemas.android.com/apk/res/
4       android:&gt;
5         android:layout_width="match_parent"
6         android:layout_height="match_parent"
7         style="@style/NumPadLayoutStyle"
8         android:rowCount="4"
9         android:columnCount="3"
10        android:background="@color/colorAccent">
11
12   <de.muffinworks.knittingapp.views.KnittingFontButton
13     style="@style/FonButtonStyle.Key"
14     android:onClick="onNumPadClick"
15     android:text="1"
16     android:layout_width="wrap_content"
17     android:layout_height="wrap_content" />
18   <de.muffinworks.knittingapp.views.KnittingFontButton
19     style="@style/FonButtonStyle.Key"
20     android:onClick="onNumPadClick"
21     android:text="2"
22     android:layout_width="wrap_content"
23     android:layout_height="wrap_content" />
24   <de.muffinworks.knittingapp.views.KnittingFontButton
25     style="@style/FonButtonStyle.Key"
26     android:onClick="onNumPadClick"
27     android:text="3"
28     android:layout_width="wrap_content"
29     android:layout_height="wrap_content" />
30   <de.muffinworks.knittingapp.views.KnittingFontButton
31     style="@style/FonButtonStyle.Key"
32     android:onClick="onNumPadClick"
33     android:text="4"
34     android:layout_width="wrap_content"
35     android:layout_height="wrap_content" />
36   <de.muffinworks.knittingapp.views.KnittingFontButton
37     style="@style/FonButtonStyle.Key"
38     android:onClick="onNumPadClick"
39     android:text="5"
40
41   <de.muffinworks.knittingapp.views.KnittingFontButton
42     style="@style/FonButtonStyle.Key"
43     android:onClick="onNumPadClick"
44     android:layout_width="wrap_content"
45     android:layout_height="wrap_content" />
46   <de.muffinworks.knittingapp.views.KnittingFontButton
47     style="@style/FonButtonStyle.Key"
48     android:onClick="onNumPadClick"
49     android:layout_width="wrap_content"
50     android:layout_height="wrap_content" />
51   <de.muffinworks.knittingapp.views.KnittingFontButton
52     style="@style/FonButtonStyle.Key"
53     android:onClick="onNumPadClick"
54     android:layout_width="wrap_content"
55     android:layout_height="wrap_content" />
56   <de.muffinworks.knittingapp.views.KnittingFontButton
57     style="@style/FonButtonStyle.Key"
58     android:onClick="onNumPadClick"
59     android:layout_width="wrap_content"
60     android:layout_height="wrap_content" />
61   <de.muffinworks.knittingapp.views.KnittingFontButton
62     style="@style/FonButtonStyle.Key"
63     android:onClick="onNumPadClick"
64     android:layout_width="wrap_content"
65     android:layout_height="wrap_content" />
66   <de.muffinworks.knittingapp.views.KnittingFontButton
67     style="@style/FonButtonStyle.Key"
68     android:onClick="onNumPadClick"
69     android:layout_width="wrap_content"
70     android:layout_height="wrap_content" />
71   <de.muffinworks.knittingapp.views.KnittingFontButton
72     style="@style/FonButtonStyle.Key"
73     android:onClick="onNumPadClick"
74     android:layout_width="wrap_content"
75     android:layout_height="wrap_content" />
76
77 </com.android.calculator2.CalculatorPadLayout>
78
79

```

Listing B.51: view_numPad.xml

```

9   <?xml version="1.0" encoding="utf-8"?>
10  <LinearLayout
11    xmlns:android="http://schemas.android.com/apk/res/
12      android:&gt;
13        android:layout_width="match_parent"
14        android:layout_height="match_parent" />
15

```

Listing B.52: view_pattern_name_input.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <merge xmlns:android="http://schemas.android.com/apk/res/
3 android>
4
5   <de.muffinworks.knittingapp.views.LineNumberTextView
6     android:cursorVisible="false"
7     style="@style/RowEditTextStyle"
8     android:elevation="4dp"
9     android:id="@+id/row_editor-line_numbers"
10    android:textSize="@dimen/
11      row_editor_default_text_size"
12    android:layout_width="10dp"
13    android:layout_height="wrap_content"
14    android:background="@color/black_20"
15    android:paddingLeft="10dp"
16    android:paddingRight="10dp"
17
18   <de.muffinworks.knittingapp.views.LinedEditorEditText
19     android:paddingRight="50dp"
20     style="@style/RowEditTextStyle"
21     android:id="@+id/row_editor_edit_text"
22     android:gravity="center-vertical-left"
23     android:textSize="@dimen/
24      row_editor_default_text_size"
25     android:layout_width="wrap_content"
26     android:layout_height="wrap_content"/>
27 </merge>

```

Listing B.53: view_row_editor.xml

Eidesstattliche Erklärung

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

.....
(Ort, Datum, Unterschrift)