Mobile Display of Knitting Patterns

Bachelor's Thesis

Bianca Ploch
540609

16 August 2016

Supervisor:
Prof. Dr. Debora Weber-Wulff

HTW Berlin
International Media and Computing (Bachelor)

Contents

Chapter 1

Introduction

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

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

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

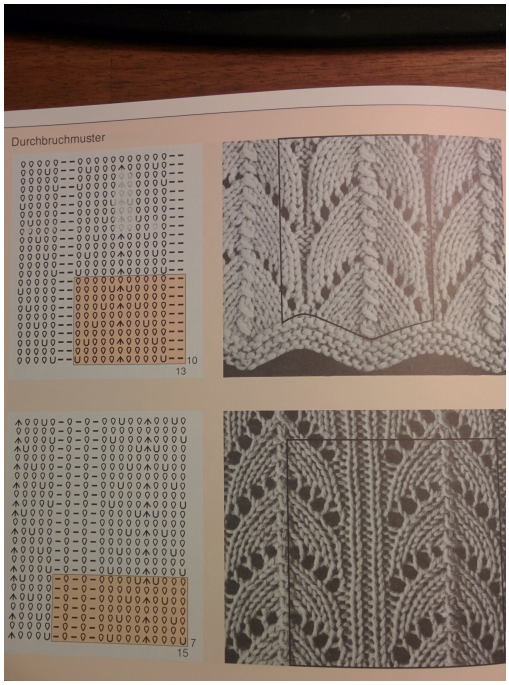


Figure 1.1: Knitting patterns and their corresponding pattern charts from Natter 1983, p142

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

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

Chapter 2

Background

2.1 Definition of Knitting Pattern and Knitting Pattern Chart

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

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

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

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

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

2.2 Comparison of Existing Solutions

2.2.1 Android apps

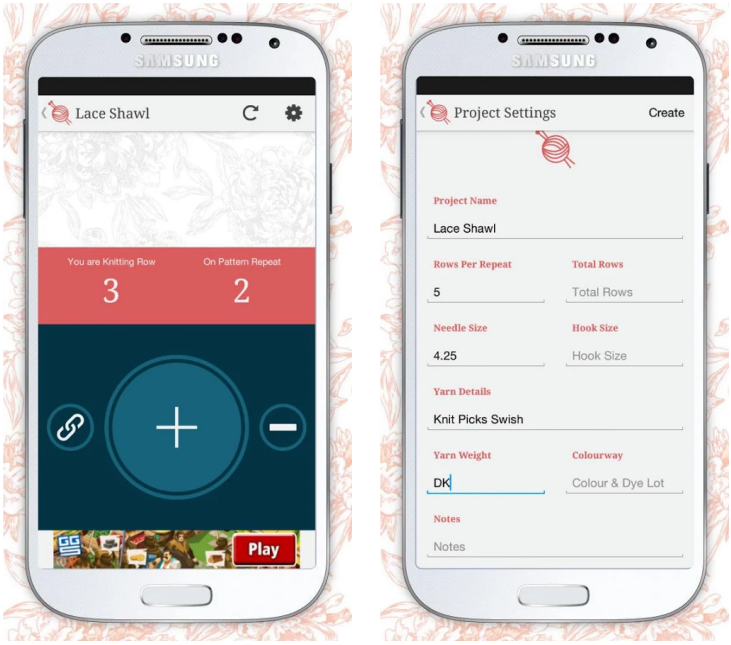
When searching for the term "knitting" in the Google Play Store, Android's official source for Google-approved applications, few results pop up. Next to a surprising amount of games about knitting, there are apps for knitting counters, knitting patterns and knitting instructions for those who wish to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

knit tink — Row Counter by Jennifer K. Warren

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

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

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



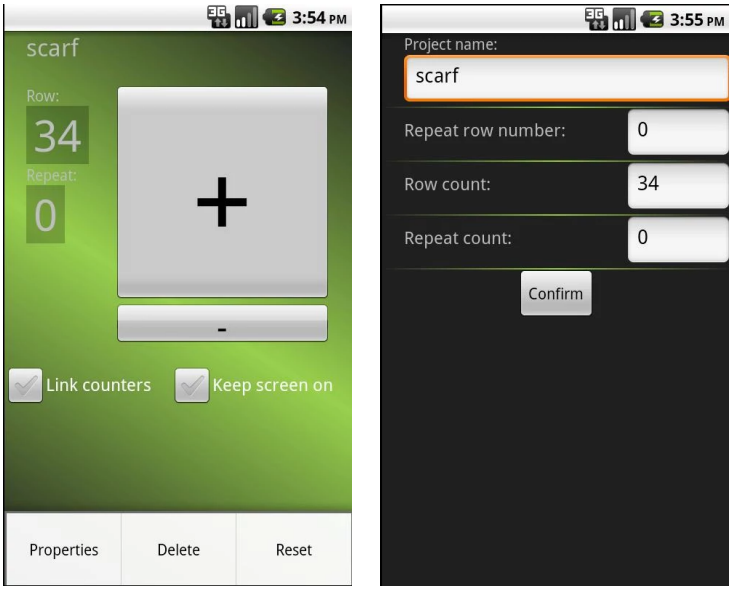
(a) Row counter (b) Project setup

Figure 2.1: Screenshots of the app knit tink

Knitting Counter by mkacki

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

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



(a) Row counter (b) Row counter setup

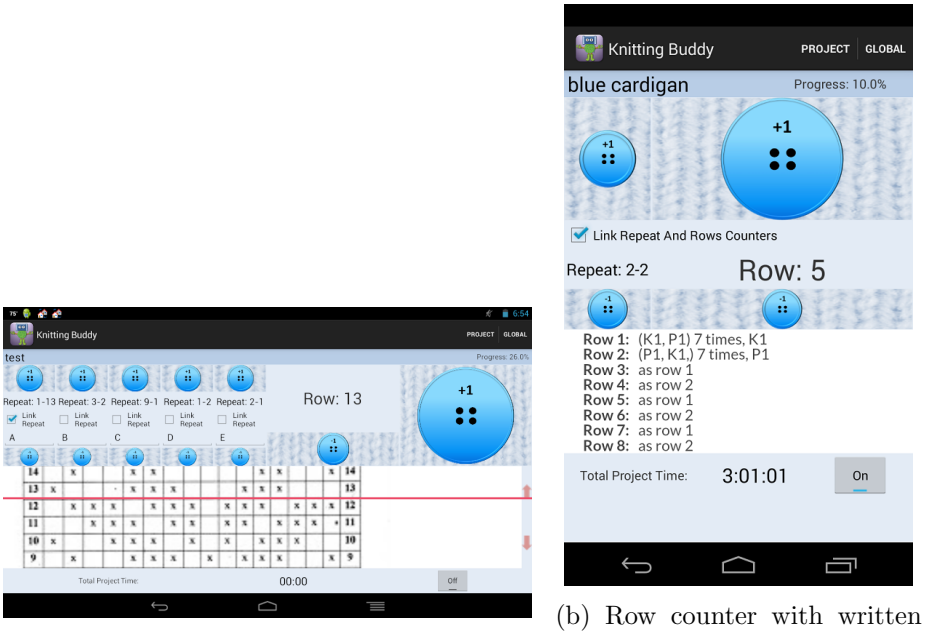
Figure 2.2: Screenshots of the app Knitting Counter

Knitting and Crochet Buddy by Colorwork Apps

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

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

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



(a) Row counter with pattern chart picture

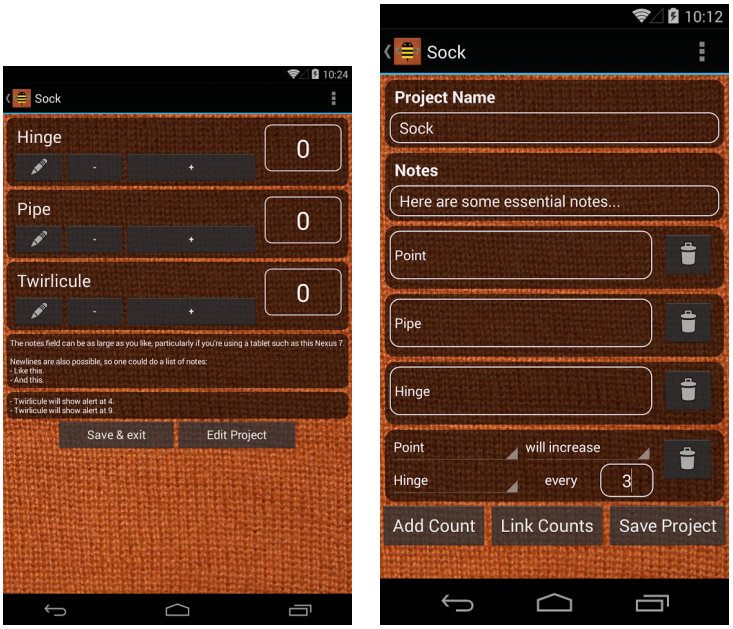
(b) Row counter with written pattern instructions

Figure 2.3: Screenshots of the app Knitting and Crochet Buddy

BeeCount knitting Counter by knirirr

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

BeeCount differs from the standard of one row counter per project in that it allows multiple counters. These counters can be for parts of the knit piece that belong to the same project, as is the case, e.g. a knitted sweater. These counters within a project can be linked together, so that increases or decreases in one counter affect the row number of another counter (figure 8). Furthermore, alerts can be set on counters to be triggered once the counter reaches a set number.



(a) Row counters (b) Row counters setup

Figure 2.4: Screenshots of the app BeeCount Knitting Counter

Knitting Chart Maker by Awesome Applications

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

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

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

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

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

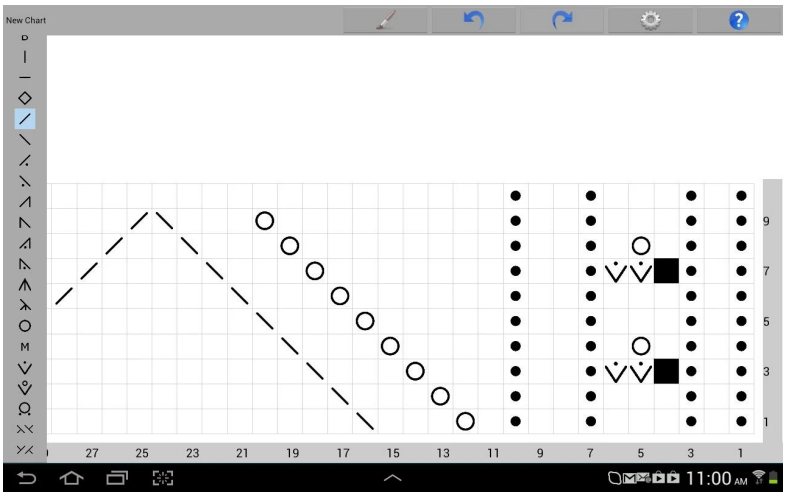


Figure 2.5: Chart editor of Knitting Chart Maker

2.2.2 Other

KnitML by Jonathan Whitall

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

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

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

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

The following KEL expression

```
Pattern {
    generalInformation
}
```

Listing 2.1: Example expression in KnitML

would result in

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

Listing 2.2: Example expression in KnitM: XML result

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

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

Chapter 3

Requirements

3.1 Functional Requirements

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

ref here

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

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

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

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

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

3.2 Non-functional Requirements

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

Chapter 4

Design

The desired outcome of this thesis will be a working Android app prototype with working 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 **??**. 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 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. While viewing that list menu entries for exporting all patterns and importing a single pattern will be available. For the import the user can choose a file on the device from a file chooser. Exporting will export files to a set directory on the publicly accessible storage of the device. A list item will consist of the pattern name, an edit, and a delete button. A click on the pattern name will open the pattern in the viewer in row format with the default dimensions of 10 columns and 10 rows. Below the list will be a button to create a new

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

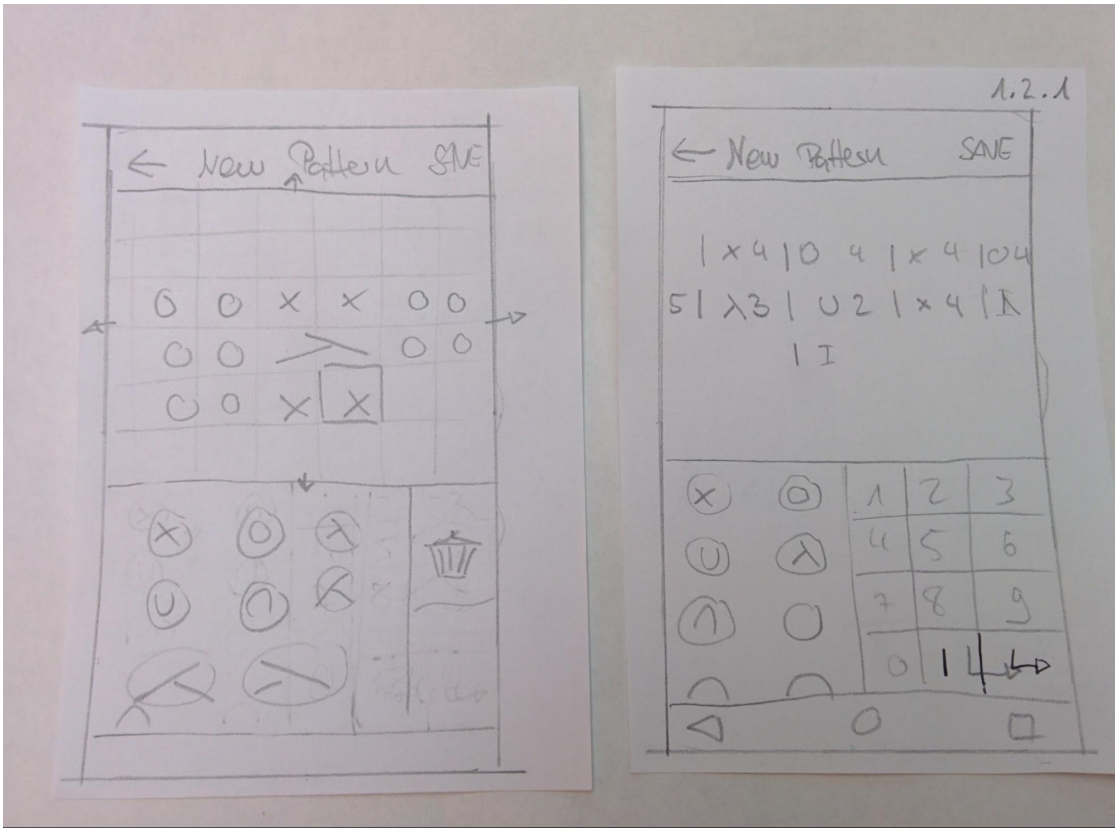


Figure 4.1: Editor screens for grid and row format with pattern name and save button in navigation bar

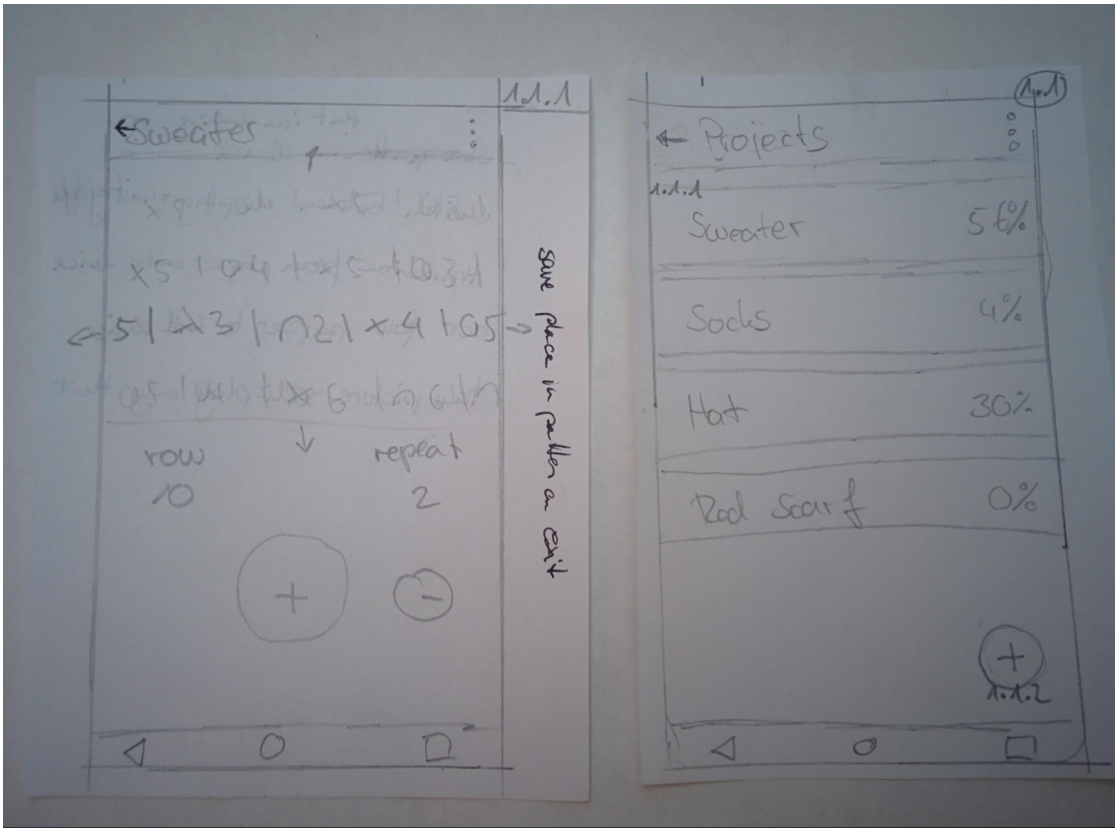


Figure 4.2: Viewer screen for row format and selection screen for stored patterns

Figure 4.3: Screens for editor and viewer without Android elements

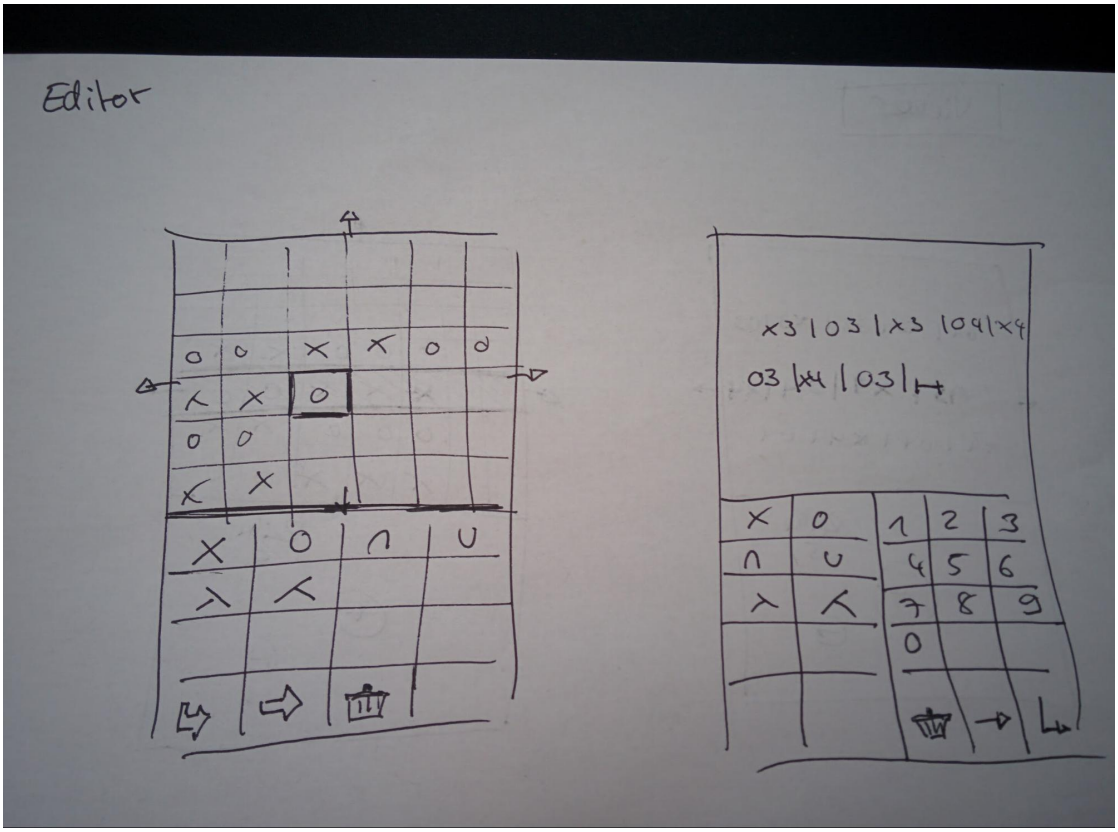


Figure 4.4: Editor screens for grid and row format

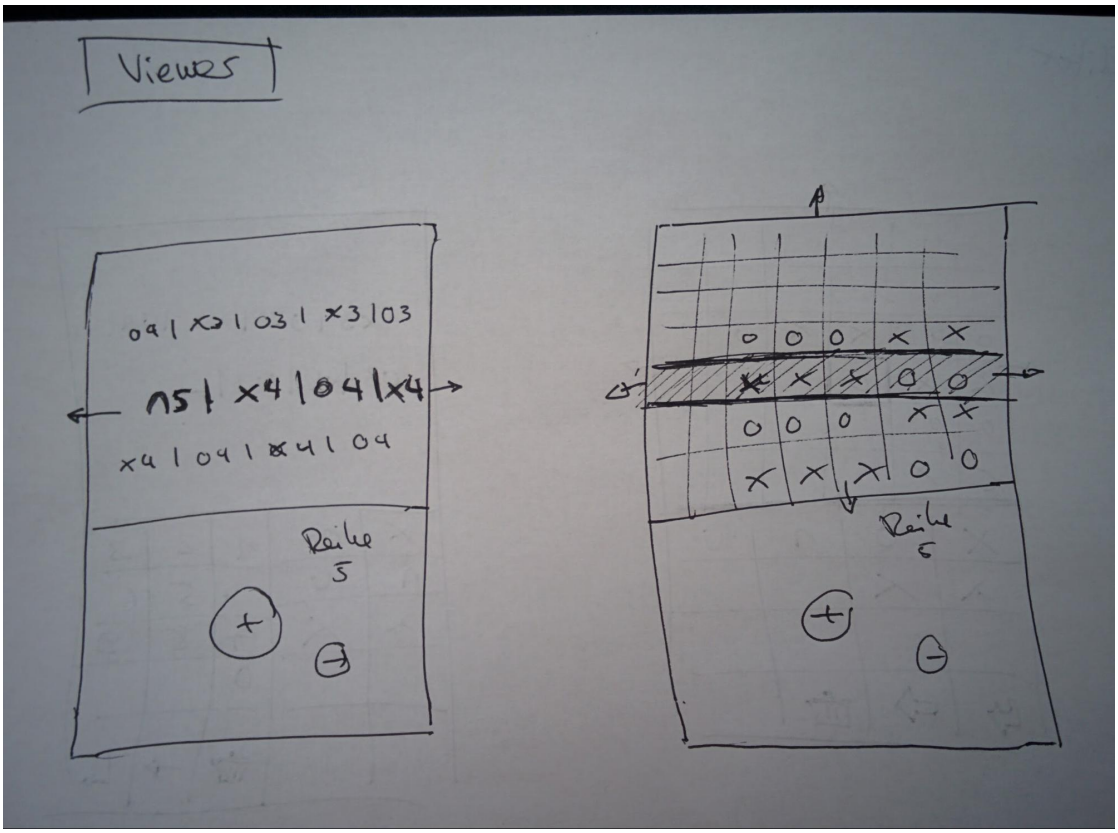


Figure 4.5: Viewer screens for grid and editor format with row counter

Chapter 5

Android Basics

5.1 The Operating System Android

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

5.2 Basic Components of an Android App

5.2.1 Activity

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

[1] `https://developer.android.com/reference/packages.html`
[2] `https://developer.android.com/training/index.html`
[3] `https://developer.android.com/guide/index.html`

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. Activities feature methods such as onCreate(), onStart(), onStop(), and onFinish(), which can be overwritten to implement logic that is executed at different points in it's life. 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. lifecycle figure

5.2.2 Actionbar

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

5.2.3 Fragment

The Fragment class ususally 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 5.1**.

A fragment's creation is always embedded in an activity (Android Developers 2016h) — forcing the fragment to pause or stop alongside its parent activity's lifecycle. Fragments can also house viewgroups and views and are intented to function as interchangeable modules, e.g. as UI modules for an app that runs on devices of varying sizes and

that wants to present the user with a dynamic UI fit to suit the screen size (see **Figure 5.2**).

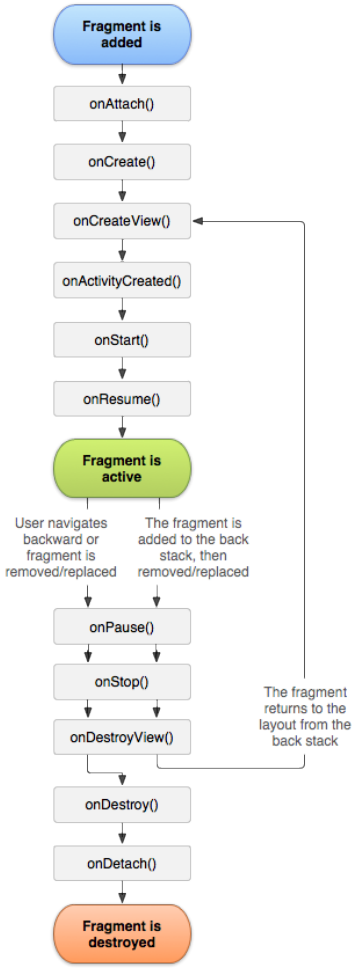


Figure 5.1: The Fragment life cycle

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 ??**). 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).

add figure dialog fragment

A good practive to enforce the implementation of a fragments callback interface is to check for its existence when the fragment is attached to the activity — example code proposed by the Android Developer guide concerning how to check for this can be seen in *Listing 5.1* (Android Developers 2016g).

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

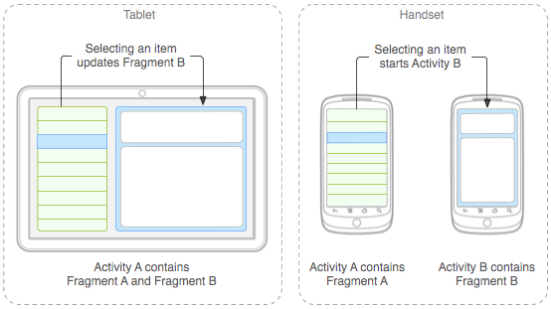


Figure 5.2: Two fragments of one activity and their layout on two different screen sizes

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 2016n). 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 an Extensible Markup Language (XML) layout file, or dynamically, from code. Android comes with plenty of view subclasses, specialized in acting as controls or displaying specific types of content, e.g. text or images. If the pre-existing views don't match a developer's needs, 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, specialized for displaying text (TextView), or a button (Button). The view class is also the basis for ViewGroups, to which the layouts, e.g. LinearLayout and RelativeLayout, belong to. Views in a window are bundled together as a tree with a layout being the top-most root. To add views to an activity or fragment the views can be declared in the corresponding XML layout or from code.

5.2.5 Storage

File storage in Android devices is separated into "internal" and "external" storage — this refers to the fact, that Android devices often times have a built-in, non-removable memory and an external, removable medium in the form of an SD or a micro SD card (Android Developers 2016f). This storage separation even exists on devices with only built-in memory — in such cases the storage is partitioned into "internal" and "external" partitions. This assures that the concept of two storages persists across all devices and API levels. The internal storage is inaccessible by the user under normal circumstances — exception to that is when the user has root privileges, e.g. on a rooted phone. This storage houses, among other things, files from apps, e.g. databases. These files are only accessible by the app that originially places them in the interal storage — neither user nor other apps can access them. Files are removed when the app they belong to is uninstalled. The external storage on the other hand is more public. Files placed here can be read and written by user and other apps and they even remain after the app they orignated from is uninstalled. When working with the external storage it is important to check that it is not currently used as Universal Serial Bus (USB) storage by a computer the device is connected to. Apps have by default read and write access to the directory they are installed in on the internal storage, but to access the external storage the app requires that the user grants the app a specific permission. This permission needs to

be declared in the app's manifest file, a XML file that every app must have. This file contains information required by the Android system to allow the app to run, such as the activities contained in the app and the permissions the app requires. Beginning in Android 6.0 (API level 23) apps targeting that Andoir version need to request and aquire dangerous permissions at run time (Android Developers 2016l), 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 2016l). Since the user can revoke permissions for apps at any given time the developer needs to take measures 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 possiblities 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 disoplaying stitch symbols, a further look into what each format entails is necessary. Using image resources

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

case scenario. The other option for displaying symbols is to create a custom True Type Font (TTF) with glyphs for stitch symbols that is applied to text in the app. This is the solution used in the prototype. It offers several advantages over using image resources. For one, when using drawables it might be necessary to include several versions of the same file to assure that they are displayed correctly on devices with different screen densities (**android˙densities**). 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 characters, 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. The open source program FontForge [1] was used creation of the custom TTF knitting font used in the protoype. The knitting font contains a selection of 16 stitch symbols whose glyphs were defined by the author herself. The glyphs and their corresponding UTF-8 characters can be found in the Appendix A: . The available symbols the knitting font offers as well as the corresponding symbol descriptions need to be defined as a inside string arrays in the Constants class in the project.

ref here?

ref here!

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 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 6.3.

The pattern 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

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

are usable by both formats. For that the class PatternParser.java is used. It converts between the array of strings used in the Pattern POJO, a two-dimensional array of strings used in the PatternGridView (6.4.1), and a single string with linefeeds used in the row format's Edittext widget (6.4.2). For a more detailed explanation of the pattern notation forms refer to the corresponding sections.

6.3 Persistent Disk Storage

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

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

cite here

ref here

The Pattern class inherits from the class Metadata which contains both a UUID which is used as a pattern's file name and the pattern name that is given by the user. When storing on disk, Pattern POJOs are converted to JSON using Gson and saved with the UUID as filename. 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 pattern 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 2016m), which is also used to position views within the constraints of cells, columns, and rows.

Android's GridView class, on the other hand, is designed with notion of displaying data. Each cell represents one data entry in a collection of data, the displaying of which is handled by an Adapter class attached to the grid view. Android ships with a specialized adapter for lists [2], 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 visible area, since the user would not be able to know the cell position then. For one, the GridView class does not support frozen cells, columns, or rows — as far as the author of this thesis was able to research. If the column numbers were to be displayed in the first row of the grid, they would move offscreen upon scrolling the grid. A possible solution to this would to use text views to act as axes to the grid and to display the column and row numbers next to it. Problematic with this approach would be the fine-tuning required to math the visuals of the text view to that of the grid. Line height, text size and the synchronization with scrolls and zooms performed on the grid would need to match perfectly. Since this does not classify as intended behavior for these views, a cohesive UI cannot be guaranteed. Furthermore, Android does not offer two dimensional scroll on any view except its WebView[3] – the GridView class natively only supports vertical scroll.

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

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

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*[4] and the corresponding training path[5] present an implementation example and were used as a guide line for the implementation of the class PatternGridView.java . This class keeps a reference to a two-dimensional array of strings which represents the pattern with its columns and rows. The grid and its axes are drawn by overriding the onDraw() method, calulating the postion of the correspoinding lines and numbers with the dimensions of the view and pre-defined, hardcoded values for cell width and margin. The current version of the PatternGridView uses a SimpleOnGestureListenerto compute two-dimensional dragging. Android defines two different scrolling types for views: dragging and flinging (Android Developers 2016e). Dragging is executed by draging a finger across the device's touchscreen and results in a moving of the view corresponding to the dragging direction and speed. This means, that no matter the velocity of the dragging gesture, the view won't keep moving once the user lifts the finger involved in the gesture. A fling on the other hand will keep moving the view with a speed and duration exponential to the velocity of the fling gesture, where the duration is calculated by introducing a friction to the scroll. This gesture can also be describe as a swiping motion performed on the touchscreen. Even after the finger has been lifted of the screen the fling continues to execute until it either runs its course or is interrupted.

reference here

code example 2d array to show columns and rows

ref here

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 Pointis saved locally to represent the offset scrolled. It is initiliazed with the value (0,0) and updated on every motion event that is recognized as dragging. The necessary calculations for these updates are done by overriding the onSCroll()method in a custom SimpleOnGestureListener. This method has access to the horizontal and vertical distance scrolled, which is substracted from the offset. This is done because the value of the distance is positive when dragging towards the point of origin (the top left corner) and negative when dragging away from it. Therefore, the canvas needs to be translated in the opposite distance. The offset is then clamped to minimum and maximum values, to ensure that the grid will never completely move offscreen:

ref docs point class

ref

[4] `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.

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

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

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

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

Similiarly to dragging, scaling is implemented with a SimpleOnScaleGestureListenerwith 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 invalidatedand re-drawn with updated pattern.

ref here

ref to method in docs here

6.4.2 Row Format

The row editor should display line numbers at the left side of the screen and keep them in that position during horizontal scrolling, so that they will not move offscreen. Additionally, it should support the standard text editor functions: text select, copy, cut, and paste. It should display text in multiple lines that do not wrap at the end of the screen, but continue offscreen until a newline is input and the content needs to be scrollable horizontally and vertically. Android's Edittext widget[6] fulfills most of these requirements. The Edittext widget inherits from the class Textview which is specialised for diplaying text content. It supports standard text editing funtions, can display multiple lines of text, and supports vertical scrolling. Unfortunately, it does not natively support text lines to continue offscreen — upon reaching the width of the widget the text is wrapped to the next line. Another problem is, that the widget always automatically triggers the showing of Android's on-screen keyboard when it receives focus, i.e. when the user taps the view to start text input.

[6] `https://developer.android.com/reference/android/widget/EditText.html`

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

ref here

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

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

Interaction with an Edittext widget will also show the on-screen keyboard: the widget's onClick() and onLongClick() listeners trigger the soft input method. To avoid this the custom widget KeyboardlessEditText[7], written by Danial Goodwin, is used in the prototype. It offers the the same functions as a native Android Edittext widget, but will suppress the showing of the on-screen keyboard.

To improve the visiblity of the lines the LinedEditorclass is subclassed from the keyboardless Edittext widget and a background is drawn behind every other line. The LinedEditor is part of the RowLinearLayout, a custom LinearLayout which houses the complete row format editing functionality. This layout will be referred to as the row editor. A LineNumberTextview, a custom Textview, is placed to the left of the 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 Textview 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.

ref

ref

ref

```
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] https://github.com/danialgoodwin/android-widget-keyboardless-edittext

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

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

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

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

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

the of a line. Instead of scrolling to the end of the edited line, the scroll will move the beginning of the next line into view.

Lasty, the line numbers do not stay visible when the row editor is scrolled horizontally. This behavior might be achieved by attaching different scrollers to the Textview 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 corresponsing with the active key being added to the pattern at the location of thetouched 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[9], a default Android component to display a collection of items in a grid with equal spacing between all items. The gridview also by default supports vertical scrolling. A grid item consists of text set on a button of the class KnittingFontButton, a custom class extending Android's own Button widget. The custom button sets the knitting font to display its string title as a knitting symbol. Set on the button are a click listener and a long click listener. The click listener toggles the state of the key and on long click the description of the symbol displayed ont he button is shown at the bottom of the screen.

refer to symbol chart here

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 apprends the corresponding string or number to the editor at the current positoin of the cursor. The enter and backspace button call the system's enter and backspace key events from Android's software keyboard and do therefore not require custom handling, but only to be forwarded to the editor.

The symbols sections is also a Gridview, although with less columns than in the grid editor. The numpad uses the CalculatorPadLayout from Rahul Parsani's Material Calculator project[10]. The CalculatorPadLayout takes a number of child views, in this

[9] `https://developer.android.com/reference/android/widget/GridView.html`
[10] `https://github.com/rahulparsani/material-calculator`

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 accommedate 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

source code ref here

The row counter and the viewer are implemented in the ViewerActivity class. The activity's layout file defines a container FrameLayout[11] for the pattern content and below that the row counter UI. On its creation the activity instantiates a PatternGridView and a RowEditorLinearLayout and sets the data of the viewed pattern on both. The view and the layout can then be switched out at runtime inside their container by adding and removing the required view whenever the user decides to switch between grid and row format. At the current version of the prototype the row format is still experiencing some issues: the line numbers are not correctly instantiated and the pattern, if larger than the screen, is not scrollable inside the viewer.

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

The actionbar contains the following action buttons

- Switch pattern formats
- Open glossary
- Scroll to current row

[11] `https://developer.android.com/reference/android/widget/FrameLayout.html`

- Export pattern
- Reset row counter
- Edit pattern

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

6.7 Editor

The class EditorActivity.javacontains, 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

ref here

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

ref
ref
ref

Editor Fragments

Each of the two editor formats has its own fragment that displays the for that format appropriate keyboard. 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 PatternListActivitythat displays all files currently indexed in the Metadata file (see section 6.3). For that Android's ListView[12] 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 the upon delete the PatternDeleteDialogFragment is shown.

ref

6.9 Glossary

Like the PatternListActivity, the GlossaryActivty also uses a ListView[13] to display the symbols and their descriptions. The symbols and their descriptions are taken from the Constants class.

[12] `https://developer.android.com/reference/android/widget/ListView.html`
[13] see footnote 12

Chapter 7

User Test

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

Ideally, user tests of the UI design are performed throughout the developement process of a product, but time permitted for only one set of user tests during the developement 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 to note wether the prototype met the expections they expressed during the interview and were 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 changed they did to the pattern in the expaned grid format. The participants did not expect the editor to support the standard editor funtionalities such as selecting, copying, cutting and pasting text. Both participants had problems when asked for the first time to edit the pattern in the grid editor: they expected the same typing behaviour from the symbol keyboard that they encountered in the row editor. It took a few tries and an explanation of the toggling mechanics to enable them to successfully use the grid editor. In both tests the participants found and used the buttons to switch editors and save the pattern without problems. The same holds true for the menu entries to rename and delete the pattern currently open in the editor. At the time of testing viewing the pattern in row format still had some bugs: larger patterns were not scrollable and the current row would not be highlighted upon increase or descrease 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 confronted with the row format on opening a pattern in the editor, the participants expected the grid editor kyeboard to behave like the one found in the row editor. They had problems understanding how to use the toggle symbols in the grid editor keyboard and were confused why nothing happend when they touched the grid. For the wish to have a symbol pre-selected was voiced, to indicate that symbols have to be selected to be set on the grid and to help the user understand that touching a cell leads to the selected symbol being set to that cell. One user also mentioned that the trash can icon

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

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

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

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

Chapter 8

Evaluation and Discussion

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

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

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

the remaining bugs is possible and only a matter of time. Known bugs that exist in the current version of the prototype are listed below:

- No limit when entering a pattern title
- Imported files aren't checked for tile type
- Pattern title not visible in viewer and editor: too many action buttons
- Editor and viewer should start with grid format instead of row format
- Drawing of the grid needs to be improved: dimensions larger than 35 cause lag on interaction
- No UI optimization for smaller screen sizes

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 handlded separately. Whether or not it might be possible to force the Edittext widget into a working solution that meets the requirements set for this thesis consistently on different devices can at this point still not be answered. More research would be needed to determine an answer to this question. It might be that the best solution for this issue is the implementation of a custom text editor.

Chapter 9

Outlook

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

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

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

hello glossary should be here):

List of Figures

Listings

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

Android Developers. *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).

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.

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).

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)