Hao Zhang

Design and Implementation of Social Event Application Based on Android

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

30 April 2019

|  |  |  |
| --- | --- | --- |
| Author  Title  Number of Pages  Date | Hao Zhang  Design and Implementation of Social Event Application Based on Android  xx pages + x appendices  30 April 2019 | |
| Degree | Bachelor of Engineering | |
| Degree Programme | Information Technology | |
| Professional Major | Mobile Solutions | |
| Instructors | Kari Salo, Head of Degree Programme | |
| Over the years, more and more tourists come to Helsinki to travel, in order to let more people know about the events happening in Helsinki area, with the help of the “City of Helsinki” organization, a social events viewing application based on Android platform is born. The main goal of this thesis is to produce a public events information platform based on Android to make it easier for people to find Activities happening in the moment or in the future and make people participate in Activities, integrate into local life and learn about Helsinki culture.  During the development process, back-end data is provided from the organization's open data which covers public data in the Helsinki region. The major application case used in this thesis is a completed social event application written in Kotlin and the specific location of the event will be marked on Google Maps. Besides, RecyclerView is wildly used in this application to display specific event information, such as date, price, event publisher and so on.  Design pattern, as an essential part of computer science, is beneficial for keeping projects architecture scalable and testable. This thesis introduces Model-View-View-Model, a design pattern encouraged for Android development. In addition, MVVM design pattern will be demonstrated along with the extracted code from the application case.  In summary, this thesis implements a social event application based on MVVM design pattern and the UI of application conforms to the “Material Design” specification. | | |
| Keywords | | Material Design, Model-View-View-Model, MVVM, Android Development, Kotlin |

Contents

List of Abbreviations

[1 Introduction 1](#_Toc3760488)

[2 Theoretical Background 1](#_Toc3760489)

[2.1 Android Fragment and Views 1](#_Toc3760490)

[2.1.1 Fragment 1](#_Toc3760491)

[2.1.2 ViewPager 2](#_Toc3760492)

[2.1.3 RecyclerView 6](#_Toc3760493)

[2.2 Android Four Main Components 7](#_Toc3760494)

[2.2.1 Activity 8](#_Toc3760495)

[2.2.2 Service 9](#_Toc3760496)

[2.2.3 Broadcast Receiver 10](#_Toc3760497)

[2.2.4 Content Provider 11](#_Toc3760498)

[3 Application Case 12](#_Toc3760499)

[3.1 Application Description 12](#_Toc3760500)

[3.2 Model View View-Model (MVVM) 14](#_Toc3760501)

[3.3 UI Design 17](#_Toc3760502)

[4 Implementation 21](#_Toc3760503)

[4.1 MVVM Implementation in Home Page 21](#_Toc3760504)

[4.1.1 Model 21](#_Toc3760505)

[4.1.2 Home Fragment View 22](#_Toc3760506)

[4.1.3 View-Model 25](#_Toc3760507)

[4.1.4 Repository 26](#_Toc3760508)

[4.2 Technology Stacks 27](#_Toc3760509)

[4.2.1 Unit Testing 27](#_Toc3760510)

[4.2.2 UI Testing 27](#_Toc3760511)

[5 Results 28](#_Toc3760512)

[6 Conclution 28](#_Toc3760513)

[7 References 28](#_Toc3760514)

[References 29](#_Toc3760515)

Appendices

Appendix 1. Title of the Appendix

Appendix 2. Title of the Appendix

List of Abbreviations

ORM Object-relational mapping. The set of rules for mapping objects in a programming language to records in a relational database, and vice versa.

DBMS Database management system. Software for maintaining, querying and updating data and metadata in a database.

[注意修改每个图片的索引]

# Introduction

Write the introduction of your thesis here. Use the Body text style for normal text. Likewise, use styles Heading 1, Heading 2, and Heading 3 for headings. As you apply the predefined styles, the text will automatically format correctly: the line spacing will be 1.5, both edges will be justified, and the text will be hyphenated.

The last line of a paragraph can be left hanging, that is, it does not have to reach the right margin.

Begin a new paragraph at the left margin, that is, do not indent the first line.

# Theoretical Background

## Android Fragment and Views

Android contains multiple views for user interface components, such as text-view, image-view, viewpager, recycler-view and so on. Since ViewPager and recycler-view are widely used in the application case of the thesis, therefore these two views will be discussed in this chapter along with android Fragment.

### Fragment

One of the powerful features added to Android 3.0 (API 11) is Fragment, which is designed to provide more dynamic and flexible UI design support for large screens such as tablets. Fragments must always be managed by the Activity , and their lifecycle is directly affected by the lifecycle of the host Activity [X].

For example, when an Activity is paused, all of its Fragments are paused as well; when the Activity is destroyed, all Fragments are also destroyed. However, when the Activity is running, each Fragment can manipulate independently, such as adding or removing them. Since each Fragment is able to designed as a modular and reusable Activity component, therefore, compared to Activity , Fragment takes up less memory space. The diagram below shows the process when a Fragment is added to an Activity :

1. Workflow of adding a Fragment to Activity . Copied from [X]

In the first step, the Activity gets a reference to the Fragment. After that, the Activity gets a reference to the View-Group where the Fragment will be rendered inside of it. Secondly, the Activity adds the Fragment and the Fragment starts to generate its view based on its independent XML file. Finally, return the created Fragment view to the Activity and the Fragment view will be inserted into the View-Group parent for displaying.

### ViewPager

ViewPager is a component of SupportV4 and it extends from the View-Group class, hence, ViewPager can be regarded as a view container. In practical working, ViewPager is most often used in conjunction with Fragment, which is a convenient way to supply and manage the lifecycle of each page [X].

1. Guide pages on Android application

ViewPager, as a view container, has multiple views inside of it which  allows the user to flip left and right through pages of data as seen in figure 2. Other than making guide pages, the most commonly used of ViewPager is to make an automatic slide show to display the hottest events or advertising exhibitions.

1. Layout of MainActivity

In the application case of this thesis, ViewPager is used to combine with Fragments as the main interface framework. As observed in Figure 3, the layout of the main screen is not very complicated. There is a tab bar at the top, and the rest of the page is filled with a ViewPager which will be filled with four Fragments.

1. Main interface framework of the application case

The ViewPager in the main page just serves as a container, mainly responsible for dynamically loading the four Fragments and each Fragment will display distinct page. As shown in the Figure 4, inside the setupViewPager() function that does the following things: instantiate a PagerAdapter.object and by invoking addFragment() function to insert 4 Fragments to the ArrayList that inside the adapter. By assigning the customized ViewPager adapter object to the adapter property of the viewpager, finally, hook up the PagerAdapter to the ViewPager,

The ViewPager class requires an adapter class to provide data to it, as observed in Figure 4, the data source used in PagerAdapter is an ArrayList to carry data items. There is an inner-class that extends the FragmentPagerAdapter abstract class and implements the getItem() method to supply instances of Fragment as new pages. The ViewPager adapter also requires to override the getCount() function, which returns the amount of pages the adapter will generate. In addition, ViewPager has built-in swipe gestures to transition through pages, and they display screen slide animations by default, so there is no need to define extra animation.

### RecyclerView

RecyclerView is one of the components in Android Lollipop 5.0, with the advent of the RecyclerView, it made its way officially. Compared with the classic ListView, the RecyclerView is more advanced, powerful and flexible enhancement of ListView especially in the aspect of displaying a scrolling list of elements based on large data sets or data that frequently changes [X].

1. Three extensions for RecyclerView

RecyclerView provides a less coupled way to implement what ListView can do, Therefore, it is widely used in the application case of this thesis, there are three main advantages of RecyclerView:

1. RecyclerView itself does not care about where and how the item is displayed, there is a object called layoutManager, as a massive enhancement brought to the RecyclerView, provides a list with different structures. In a ListView, the only type of view available is the vertical ListView. There is no official way to even implement a horizontal ListView. However in RecyclerView,it offers many customization options, such as LinearLayoutManager, StaggeredLayoutManager and GridLayoutManager. If these above layout does not suit project’s need, developers are uble to create their own layout structure by extending the RecyclerView.LayoutManager abstract class.
2. RecyclerView itself does not care about the effect of item addition and deletion animation. The tranditional ListView is lacking in support of good animations, but the RecyclerView uses an animator to alter its appearence. Developers can define their own animator object by extending RecyclerView.ItemAnimator. With the RecyclerView.ItemAnimator class, animating the views becomes easier and intuitive. On the other hand, animation makes the display of list elements more in line with Material Design specifications.
3. RecyclerView itself just care about how to take the view, the views in the list are represented by view holder objects. These objects are instances of a class you define by extending RecyclerView.ViewHolder. Each view holder is in charge of displaying a single item with a view. In a ListView, it was recommended to use the ViewHolder pattern but it was never a compulsion. In case of RecyclerView, this is mandatory using the RecyclerView.ViewHolder class. When implementing a RecyclerView this class is used to define a ViewHolder object which is used by the adapter to bind ViewHolder with a position. In this way, RecyclerView avoids a heavy operation of finding views by ids every time.

## Android Four Main Components

The Android application is mainly composed of four components: Activity, Service, Broadcast Receiver and Content Provider. The development of Android application is inseparable from these four major components.

1. Relationship between Android four major components

In the process of implementing the application case, with the help of the four components. Functionality such as map display, user location monitoring, components communication and so on are realized. The relationships between components are shown in Figure 6. Each component will be specifically introduced in the following paragraphs.

### Activity

The most frequently used in Android is the Activity component, which is one of the most basic modules in Android. Activity takes care of generating a window for placing UI with setContentView(View) method. Activities are usually presented to the user as a full screen window. These interfaces basically belong to or depend on the Activity . In the android program, the Activity generally represents a screen of the mobile phone. If compare a mobile phone to a web browser, the Activity is equivalent to a web page. Similar to the Input, H1, Span and other elements contained in a HTML page, Button, Checkbox and other controls can be added in the Activity. From another perspective, the concept of Activity and the concept of the web page is similar to each other.

Generally, an android application is composed of multiple Activities, and multiple Activities can jump between each other. For example, pressing a Button may jump to other Activities. When a new Activity is opened, the previous Activity is placed in a paused state and pushed into the top of the Activity stack. The user can return to the previously opened Activity by the back button on the phone. There is a choice of Activities that are not necessary to keep, thus, those Activities can be selectively removed from the Activity stack. A little different from web page jumping, jump between Activities may return values. For example, from Activity A to Activity B, then when Activity B finishes running, it may give Activity A a return value. As shown in Figure 6, Launch an Activity for which the previous Acticity would like a result when it finished.

1. Start a Activity and wait for a result

In addition, the Activity component is designed to have a system control in the form of a lifecycle. The Activity mainly has: onCreate, onStart, onResume, onPause, onStop, onDestroy, and onRestart lifecycle methods. When developers design the functionality of an application, they only need to match the lifecycle according to the business to determine what needs to be done at different lifecycle methods.

### Service

Service is a solution for running programs in the background process. It is very suitable for tasks that do not need to interact with users but also run for a long time. The operation of the service does not depend on any user interface, but can be used to interact with other components, even if the current application is switched to the background, or the user opens another application, The service is still up and running because the service is not running in a separate process, but rather on the application process in which the service was created. When an application process is killed, all services that depend on it will stop running.

The service can be used in applications with multiple occasions, such as detecting changes in the file on the SD card, or recording changes in the location of your geographic information in the background. The commonality of the above cases is that these operations do not require an interface display through the service.A typical case is music application in which an Activity starts a service running on user interaction, with the service probably downloading music from the web server. The user can continue to interact with the Activity while the service runs since it executes in the background.

### Broadcast Receiver

Broadcast Receiver is a component that can be regarded as a messaging system across applications and outside of the normal user flow. Similar to the Service component, it has no user interface. The principle behind of Broadcast Receiver is kind of like publish-subscribe pattern in Youtube. In youtube, one user can subscribe to multiple videos to get an update reminder. Similarly, the user can also post videos and be subscribed to by others.

1. Example of detecting battery change using Broadcast Receiver

Back to adnroid, after the broadcast receiver is defined by one application, other applications call it according to its defined rules and send a broadcast to it. After receiving the broadcast, the received data can start an Activity or a Service to start the follow-up function. In additon, an application can define multiple broadcast recipients, either dynamically by inheriting from the BroadcastReceiver class as illustrated in Figure 7 or statically registered in the AndroidManifest file.

### Content Provider

In Android, the protection of data is very strict. Except for the data placed in the SD card, the database, files and other contents held by an application are not allowed to be directly accessed. Android certainly does not really make every application an "island", it has a ”window” for all applications to interact with data from other application, this ”window” is the Content Provider.

1. illustration of migrating content provider storage [X]

The Content Provider is an access plan for third-party application data provided by Android. Content providers are used to share a given data set of an application to other applications, and other applications can obtain data from a content provider. The relational database SQLite provided in Android system will create its own data set for each application level, and only the content provider can share data between each application. Content Provider is able to modify application data store implementation without affecting other existing applications that rely on accessing data. In this case, only Content Provider is affected, not the application that accesses it. For instance, the SQLite database can be replaced with an alternate storage, as shown in Figure 7 [X].

# Application Case

## Application Description

The application case of this paper is the result of my participation in an innovation project, which is a joint event organized by Helsinki Metropolia University of Applied Science and City of Helsinki. The City of Helsinki organization supplies backend database called “Open Data” and API interface to provide data for this project and our project team is mainly responsible for front-end development based on the Android platform.

The core idea of the application is to provide a platform for people living in Helsinki or traveling in Helsinki to find real-time public events. The project adopts the MVVM framework design, and the client interface design follows the specification of material design, MVVM architecture and Material Design are further discussed in the following sections. Moreover, the client is programmed with Kotlin and uses some third-party libraries, such as Retrofit, Google-Map and so on. The main functions of the project are as follows:

1. List upcoming events and display specific event information such as time, location, price, publisher, etc.
2. Show the specific location of the event as a marker on the map
3. Filter result by category, age, price, postal code, date and so on
4. Create peronal event and post it to Open Data
5. Display multiple itineraries to the event location
6. Part of the logic diagram of the project

As shown in Figure 10, the main interface MainActivity uses a ViewPager as a container to load different Fragments. The HomeFragment is mainly used to display activities of a specific theme. The ongoing event fragment is used to display events that will take place in the Helsinki area in the coming week. Moreover, by clicking on each activity card will get data by requesting open data and jump to the event details page. In the event details page, users can click the map card shown in Figure 15 to get the specified route plan through the API of Helsinki public transportation. Similarly, users are able to post an event to the public. SettingsFragment is the place where users can change the theme, language and other options. It should be noted that the above figure is only a part of the overall logic diagram. It mainly lists the usage of ViewPager, RecyclerView, Activity and other components that listed in Chaper 2, since the thesis focuses on the implementation of the MVVM framework in Android applications, thus, Manager classes Implementation , such as networking class, data persistence class and so on, will not be covered in this thesis.

## Model View View-Model (MVVM)

In the process of large-scale software system development, if developers do not pay attention to the architecture of the program, the modular design of the code and the decoupling of the function modules, this may lead to some undetectable and difficult to locate errors. Especially these undetectable problems occur when the program reaches a certain scale, Due to the high degree of coupling between functions and services, it can lead to difficulties in reconstruction, finally, it can only be achieved by redesigning the software. In order to improve development efficiency and lay the foundation for future software maintenance. it is crucial to define a reasonable program architecture in the early development process.

1. The relationships between View, View-Model and Model (Copied from David Britch (2017) [X])

The MVVM framework model was first proposed by Microsoft and applied in software development. With the rise of mobile development, MVVM is also encouraged by Google, as shown in Figure 11, MVVM is similar to the well-known MVC pattern to some extent because Model layer and View layer are relatively identical. The only difference is C (controller) and VM (view model).

1. View

The View layer is only responsible for UI-related work, UI and data are strictly separated. It does not perform logical processing and update the UI in the Activity or Fragment. UI renew is implemented by changing the data source in the View-Model layer. Simply put: In View layer, the main role of View is informing the View-Model about the user’s actions

1. Vew-Model

The View-Model layer does exactly the opposite of the View layer. View-Model only does things related to business logic and data source. It does not do anything related to UI. The View-Model layer does not hold any reference to the UI element and it is unable to update the UI by reference to the UI element. Simply put: View-Model is not tied to the view, the main role of the View-Model layer is to wrap the model and arrange observable data needed by the view.

1. Model

The biggest feature of the Model layer is that it is assigned the responsibility of data acquisition. View-Model is used with Model to capture and save data. The Model provides a data acquisition interface for the ViewModel to invoke, and through data transformation and manipulation and finally mapping to the properties of a UI element of the View layer.

1. Google recommended application architecture (Copied from Google Developers guides [X])

However, In the actual development process, a Repository layer is usually added between the Model layer and the View-Model layer as shown in Figure 1, this Repository layer is equivalent to a mediator between different data sources, such as persistence models, web services, and caches. The main role of the Repository module is to handle data operations, they offer a clean API so that the rest of the application can easily retrieve this data and they aslo know where to get the data from and what API calls to make when data is updated. Moreover, The structure diagram shown in Figure 1 is also the official application structure recommended by google [X].

in summary, each component depends only on the component that one level below it. For instance, Activities and Fragments only depend on the View-Model layer and the Repository layer is the only module that depends on many data sources; in this example, the repository relies on a persistent data model and a remote back-end data source [X]. Both the unit test of the UI and the unit test of the business logic are low-coupling. This greatly improves the testability of the application. In addition, due to the low coupling of the MVVM framework, team development is more convenient, such as one developer handling business and data, and another developer responsible for specialized UI processing.

## UI Design

Application’s interface design is kind of like the industrial design in industrial products, which is an important selling point of the product. The criteria for verifying an interface are neither the opinion of a project development team leader nor the result of a project member's vote, but the user's feelings. An application with a reasonable and pretty interface will not only bring a comfortable visual enjoyment, but also bring people closer to the product. The UI design of this thesis’s application case uses Material Design as guideline to create a consistent interface and user experience.

In the period of disorder, Android is full of the style of freedom, since Google did not impose any restrictions about design guidelines. Developers can arbitrarily upload applications designed with their own ideas to the store without review. At that disorder period, Android is like a wasteland opened by Google that everyone can use at will. The products of that period had no user experience, and users were forced to adapt to different interaction styles.

However, over the years, In the rapid development of mobile development, interface design is getting more and more attention, especially after Google released Material Design on Google I/O conference in 2014. Google rethinks the user experience on the Android platform, trying to bring the experience and physical feedback of the real world to the screen. At the same time, remove the impurities and randomness in reality, retain its most primitive and pure form, spatial relationship, change and transition. Finaly, restore the most realistic experience and achieve a simple and intuitive effect. [X]

1. Cards design used in Trello [X])

Card, as one of the component in Material Design, borrows the features and concepts of cards in the real world, it has been widely used in mobile development. For example, Facebook's feed uses cards with infinite scroll loading to carry a quick preview of events. In addition, Trello's task list also uses a card design, which is very helpful for users to manage different tasks as shown in Figure 12.

1. Structure of a single card component in Material Design [X])

Usually the cards in the UI are rectangular, which carries different elements such as images, text, links, buttons, and so on. Different elements perform their duties in different cards as observed in Figure 13. Due to the independence of each card, this makes the information more portable and easier to share. Moreover, gestures are the main interaction in the mobile design, interesting gestures and interactions can create a fun and enjoyable experience with cards. For example, choose whether you like the content in the card by swiping the card left or right, organize the card with a long press click,, etc.

1. Sample of map card in the application case

In the application case of this paper, the card layout design is used in the project. Meanwhile the card is used together with RecyclerView, each card is used as an item in the RecyclerView to form a Facebook-like event stream and timeline. Different content is loaded into different cards, for example, some cards display specific information about the activity, while some cards are used to load a map to display the location information of the event as shown in Figure 14.

# Implementation

## MVVM Implementation in Home Page

Since the project is developed using the MVVM framework, therefore, there are multiple MVVM blocks in the application, for study purpose, only one MVVM block that related event details will get a concrete explanation as the study case, this block includes the following classes: EventDetailsData, EventDetailsActivity, and EventDetailsViewModel which represent Model, View, ViewModel respectively. For those classes that are not related to MVVM, such as management classes, tool classes, etc., will not be mentioned in this paper.

### Model

The data displayed on the home page are encapsulated into multiple data models. In the event model, as shown in the Figure 16. since there are more than 30 fields in event data model, only a few important parameters are displayed in the Figure.

1. Sample of event model used in Home Page

The meaning of each field in the event model is self-explanatory. Notice that the location field represents an object containing information about the location. There is a location id in the location object. The location id can be used to obtain the location details, similar to the joint query in the database.

1. The usage of Model class in Retrofit

The data retrieved from the backend server is returned to the client as JSON string format. As for Model classes, the main role in the entire application is to map values of the JSON. In the project, a third-party library called Retrofit is used to process the network request. As shown in the Figure 17, when building a Retrofit object, a GsonConvertor object needs to be passed in to initialize the Retrofit object. Retrofit will use the converter library chosen to handle the deserialization of data from an object. In the NetworkingServices interface, there is a loadEventsByPageNum fuction which return value is a parameterized Call<T> object to perform GET HTTP request, in this example, the return value is Call<List<EventModel>>. EventModel class is used to map the response JSON KEY parameters to their respective variables, for example, the value of key endTime will be mapped into endTime property in the Event Model class

### Home Fragment View

The recommended way to communicate between View and View-Model is the observer pattern, which is available in an observable way provided by LiveData or other libraries. In the project, LiveData was choose to build a bridge between these two layers, LiveData is an observer wrapper over a model data that is lifecycle aware, for example: LiveData<MutableList<EventModel>>, a list of the Event Model classes is wrapped as observable data as illustrated in Figure 18.

1. Observer logic diagram with LiveData

UI of Home Fragment is driven by the data in the Event Model class. According to MVVM principle, View layer relies on the ViewModel layer, thus, HomeFragment holds a reference of HomeFragmentViewModel as shown in Figure 19 and this reference is used to subscribe. In the subscribeObservers funtion, LiveData was exposed by ViewModel and observed by View, if LiveData changes, View will be notified and update itself with new data. This avoids the developer taking the time to re-update the data by findViewById method for the View layer, which simplifies operations and improves development and testing efficiency.

1. Home Fragment Controller

In addition, HomeFragmentViewModel exposes two ViewModels, one is related to Event Model, another one is in the way like LiveData<Boolean> which is used for detecting data state and determine whether the data is loaded successfully. For example, in Figure 19, A scroll listener is added to the RecyclerView, which triggers the searchNextPage method when RecyclerView reaches the bottom. Then, a ProgressBar will appear when pulling up to load the next page and it will disappear when next page is loaded.

1. Observer logic diagram with LiveData

Another point to note is that LiveData knows about the View layer’s state, since “this” as a parameter in the observer function is set to current LiveData’s lifecycle owner, in other words, HomeFragment is in charge of both LiveData<MutableList<EventModel>> and LiveData<Boolean>, therefore, LiveData will not be triggered if its lifecycle owner is destroyed as illustrated in Figure 20.

### View-Model

ViewModel is used to store and manage UI-related data in a life-cycle-aware manner. It allows data to survive without loss when the configuration changes. For example, the most common one is when the Activity is switched between horizontal and vertical, this Activity will be destroyed and recall the onCreate() function, but LiveData is not be triggeted, thus, data survives.

1. Sample of View-Model for HomeFragment

As observed in Figure 21, the HomeFragmentViewModel class extends from the ViewModel class, and there is a LiveData inside it and ViewModel has no view’s references, so there is no memory leak and no need to handle lifecycle events, such as, unsubscribe observers in onDestory() lifecycle methods.

### Repository

Repository is responsible for obtaining data from different data sources. for example, The data can come from the network (Retrofit + OKHttp), the local Database (Room), or cache, etc. When ViewModel gets data from the Repository, ViewModel is no need to pay attention to which data source the data comes from.

1. Fetch data from server in Repository

As observed from Figure 21, This EventRepository class is defined using the “object” keyword in Kotlin, making it a singleton pattern, so that Kotlin will not create multiple Repository instance objects when fetching data. In this EventRepository, there is a method called getEventModels which is used to fetch data from server by calling the HTTP request. After successfully acquiring the data, the value of responseDataSet variable will be reassigned and return it to the View-Model. At the same time, since View layer relies on the View-Model, therefore, View layer will get notification of data changes.

## Technology Stacks

### Unit Testing

### UI Testing

The app with the Android operating system is developed in the Java language, which is an object-oriented programming language. Since Java was created, it has been honed for many years and the use of many developers, feedback, maintenance, Java is now very mature, with many excellent features compared with other languages, such as Java is a simple Object-oriented languages no longer use C++'s multi-inheritance, operator overloading and other language features that are difficult to understand. Java also has distributed features. Java supports network programming. Java guarantees through language-level exception trapping mechanism. Its robustness, the Java language is also portable, because the Java language provides the specified basic type data, the Java language provides multi-threading at the same time, and provides a complete set of mechanisms for multi-threading to ensure the correctness of the data. With the further development and improvement of Java, the performance of Java is getting higher and higher on different platforms, and the speed of running on some platforms has not lost C++.

# Results

# Conclution

# References

References

1. Official documentation of Fragment [online] Google Corporation; URL: [https://developer.android.com/guide/components/Fragments](https://developer.android.com/guide/components/fragments) Accessed 23 February 2019
2. Anupam Chugh. Android Fragment Lifecycle [online]; URL: [https://www.journaldev.com/9266/android-Fragment-lifecycle](https://www.journaldev.com/9266/android-fragment-lifecycle) Accessed 22 February 2019
3. Official documentation of ViewPager [online] Google Corporation; URL: <https://developer.android.com/reference/android/support/v4/view/ViewPager> Accessed 23 February 2019
4. Create a List with RecyclerView [online] Google Corporation; URL: <https://developer.android.com/guide/topics/ui/layout/recyclerview> Accessed 23 February 2019
5. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019
6. The Model-View-View-Model Pattern; 2017. [online] Microsoft; URL: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm> Accessed 23 February 2019
7. Principles of Material Design [online] Google Corporation; URL: <https://material.io/design/introduction/#principles> Accessed 23 February 2019
8. Cards in Material Design [online] Google Corporation; URL: <https://material.io/design/components/cards.html#anatomy> Accessed 23 February 2019
9. Amit Agarwal. How to Get Things Done with Trello; 2015 [online] URL: <https://www.labnol.org/internet/trello-basics-getting-started/29044/> Accessed 23 February 2019`
10. Model View ViewModel MVVM Android Example [online] URL: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel> Accessed 23 February 2019
11. Guide to app architecture [online] Google Corporation; URL: <https://developer.android.com/jetpack/docs/guide> Accessed 23 February 2019
12. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019
13. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019
14. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019
15. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019
16. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019
17. Content providers [online] Google Corporation; URL: <https://developer.android.com/guide/topics/providers/content-providers.html> Accessed 23 February 2019