

Master's thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science and Engineering

Aspect-oriented user interface design for Android applications

Bc. Jiří Šebek

Study program: Electrical Engineering and Computer Science

Branch: Information Technology

March 2013

Supervisor: Ing. Tomáš Černý

Acknowledgement / Declaration

I thank Ing. Tomáš Černý, the supervisor of the diploma thesis for his valuable advice and suggestions, which were very helpful. Furthermore i thank my family for support during studies.

I declare, that i have done assigned the diploma thesis alone led by supervisor. I used only literature, that is listed in work. Furthermore i declare, that i have no objections against lending or making public of my diploma thesis or it's part with agreement of department.

In Prague 9. 5. 2014

.....

Abstrakt / Abstract

Diplomová práce se zabývá úlohou vytvoření efektivního android frameworku, pomocí kterého mohou vývojáři vytvořit android aplikace velmi jednoduše v krátkém časovém intervalu pomocí aspektového návrhu aplikací. Při řešení úlohy bylo využito jazyka Java s IDE eclipse a s Android SDK na operačním systému Windows 7. Bylo navrženo řešení, které umožňuje řešit jednotlivé aspekty, jako jsou bezpečnost, rozvržení prvků, validace vstupu, provázání dat a prezentace, nezávisle na sobě. Navržené řešení bylo porovnáno s klasickým vývojem mobilních aplikací a také s frameworkem Aspect Faces, který aspektový přístup také využívá, ale je určen pro Java EE aplikace. Jednotlivé dílčí aspekty frameworku byly otestovány a bylo prokázáno, že framework je efektivní v následujících oblastech: nezpomaluje danou aplikaci ale zrychluje ji, zpřehledňuje kód, urychluje vývoj a snižuje počet řádků kódu, který musí vývojář napsat.

Klíčová slova: aspektově orientovaný přístup, design řízený aspekty, přístup řízený inspekcí entity, runtime aspektový model, snížení úsilí při vývoji a údržbě

This diploma thesis deals with the creation of an effective android framework that will allow a developer to create android applications very easily in a short period of time with aspect approach. Through the solving of the given task the Java language was used with the IDE eclipse and with the Android SDK on operation system Windows 7. The solution was designed which enable to deal with separated aspects like security, layout, input validation, data binding and presentation independently. The specified solution was compared with conventional approach of developing mobile applications and also with framework Aspect Faces that is also using aspect oriented approach but it is designed for Java EE applications. Each aspect of framework was tested and it was proven that framework is effective in the following domains: it is not slowing down the application that is using designed framework but it is even faster, clear the code to be more readable, it is making develop faster and lower the number of code lines that developer has to write down.


Keywords: aspect oriented approach, aspect driven design, entity inspection based approach, runtime aspect model, reduced maintenance and development efforts

Contents /

1 Introduction	1	10.1 The pick of operation sys-	
2 Background	2	tem (OS) and IDE	42
2.1 Conventional approach	3	10.2 Instalation steps	42
2.2 Aspect oriented approach	4	11 Conclusion	43
2.3 Applications developed in		A Specification	46
Java for Android	5	B Symbols	47
2.4 Life cycle of Android ap-		C Code examples	48
plications	7	D References	49
3 Related work	10	E Content of attached CD	51
4 Framework requirements	12		
5 Analysis and design of the			
framework	15		
5.1 Analytic model of classes	15		
5.2 Sequence diagram of			
frameworks lifetime	16		
6 Implementation	18		
6.1 Directory structure	18		
6.2 The cache of framework	20		
6.3 Data presentation	22		
6.4 Data binding	25		
6.5 Input validation	25		
6.6 Layout	29		
6.7 Security	31		
6.8 Error messages	32		
7 Comparison of aspect ori-			
ented approach and con-			
ventional approach for An-			
droid platform	35		
8 Comparison of aspect ori-			
ented programming (AOP)			
for Android platform and			
Java EE	39		
9 Table of regresive tests	41		
10 Instalation	42		

Tables / Figures

6.1. Asymptotic time complexity of hashMap21	2.1. Distribution of all OS for mobiles 2
7.1. Comparison of AOP and conventional approach.....36	2.2. The structure of conventional android application 3
7.2. Table of launching times36	2.3. Xml generated view vs programmatically generated view 4
9.1. Regresive tests41	2.4. Creating and executing Java program..... 5
	2.5. Creating and executing java program for the desktop and for the Android 6
	2.6. Structure of Android 7
	2.7. Life cycle of android activity.. 9
	4.1. Distribution of Adroid versions13
	5.1. Analytic model of classes.....16
	5.2. Sequence diagram of framework17
	6.1. Directory structure of framework.....18
	6.2. Example how resources works19
	6.3. How hashMap works21
	6.4. The result of presentation....24
	6.5. The result of presentation with invoked software keyboards.....24
	6.6. Input validation before view is rendered26
	6.7. Input validation while the is changed.....27
	6.8. Data succesfully passed validation28



6.9.	Templates of basic two layouts	30
6.10.	The real view of basic two layouts	31
6.11.	The example of usage security	32
6.12.	The options of error messages	34
7.1.	Formula of standard deviation	37
7.2.	The launching times of application	38

Chapter 1

Introduction

The main task of this diploma thesis is to create an android framework that will allow a developer to create android applications with minimal effort in a short period of time with the aspect approach.

The aspect oriented approach in a software development means that we are focusing on each divided aspect. These aspects are: security, layout, input validation, data binding and presentation. In the conventional approach, we are mixing a code of all these aspects together as one big code. As a result of the aspect approach, the developer can write a less amount of code which is reusable. We also avoid a spaghetti code, a redundancy of code and other bad habits in the programming.

In the first part, the background research of the aspect oriented approach in the software development is described. This chapter also reveals the nature of many benefits of the aspect oriented approach. The second part contains particular framework requirements on developer's computer. An analysis and a design of the framework are included in the third part of this diploma thesis. All of later shown diagrams were made in the Enterprise Architect software. The fourth part contains the proposed framework's implementation. The final implementation was designed according to the modern OOP requirements. As a result, the proposed framework provides easy scalability and also fast response time. The implementation contains all of the mentioned aspects. In the fifth part, the comparison with the conventional approach of developing android applications is described. There is also a list of cons and benefits. The sixth part is dedicated to the installation of framework. In the conclusion, there are stated achieved results and the discussion is provided.

Chapter 2

Background

Within all operation systems (OS) for a mobile device, the Android is the most expanded as shown in Figure 2.1. Because of that, the most desirable applications are targeted for Android. As you can see from Figure 2.1, the ratio of applications targeted for the Android OS for mobile device is steadily growing up. This is the reason why developers cannot omit the Android in their analysis.

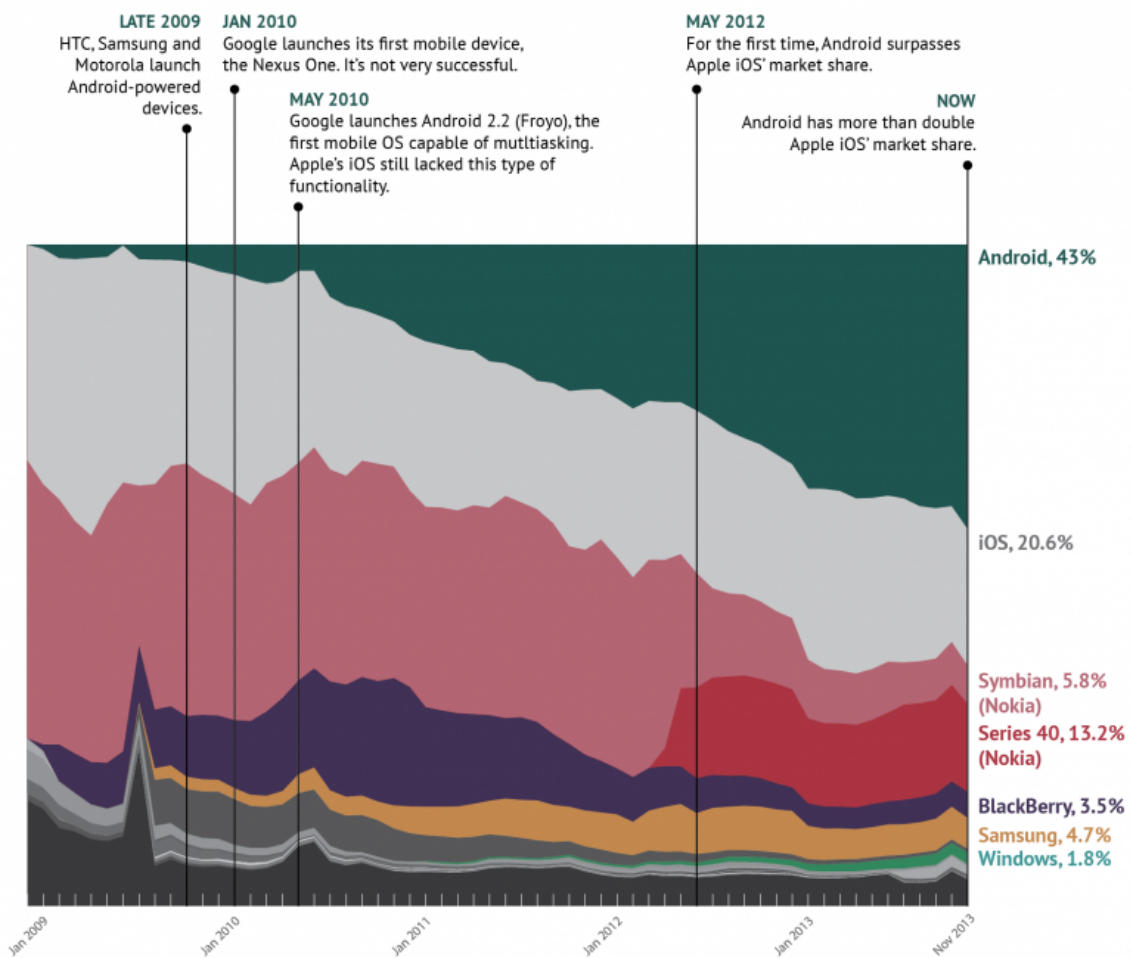


Figure 2.1. Distribution of all OS for mobiles (undertook from reference 6 in Appendix D)

2.1 Conventional approach

In the conventional object oriented approach, we firstly install SDK and a plugin for some IDE and then we create a project. We can now see the structure of the created project in Figure 2.2.

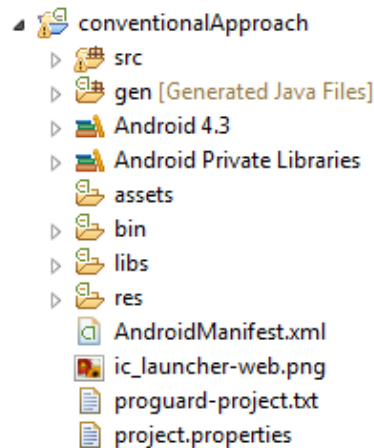


Figure 2.2. The structure of conventional android application

In android applications, there are two parts of every screen. Every screen in android is called Activity. The first part is java class that extends Activity where you can place your logic and you have to connect your java class to view. That view is second part and that can be done by xml file or programmatically. In the folder named /src, there are java files and in the folder /res/layout, there are xml files. Every new activity has to be registered in AndroidManifest.xml. In this file, there is also a global class, icons, filters, an internet permission etc. In Figure 2.3, you can see the difference between these two implementations of the view part. Usually, xml choice is better because you can separate, at least, layout from remaining code. In Figure 2.3, you can also see that in xml part, you can switch the tab to see the layout in any IDE. Sometimes it is really helpful.

```

View defaultlayout = new RelativeLayout(context);
((RelativeLayout)layout).setPadding(20, 20, 20, 20);
defaultlayout.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
TextView tv_header = new TextView(context);
tv_header.setText("Your data was saved");
tv_header.setTextSize(16);
tv_header.setGravity(Gravity.CENTER);
((ViewGroup) layout).addView(tv_header);

setContentView(layout);

```

VS

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Your data was saved.." />

</RelativeLayout>

```

Graphical Layout activity_main.xml

Figure 2.3. Xml generated view vs programmatically generated view

2.2 Aspect oriented approach

In the aspect oriented approach, we wanted to separate all mentioned aspects resulting in a good code. In OOP designs, we use classes to describe only instance and its attributes. That is representation for data only. Therefore, the best way to do aspect way approach is to add these additional pieces of information to the same representation (class). This is called Rich Entity Aspect/Audit Design (READ) [1,2] in Appendix D. Above any instance, attribute or method you can place annotation to add another information. By this procedure, we can add all required aspects.

2.3 Applications developed in Java for Android

The basic steps for developing Java applications are:

- Write the program in Java
- Compile the source code
- Run the program

The first step is to write a code and to save it in a file with extension `.java`. Then, we compile this file, which contains the source code, with compiler. The default java compiler is called `javac`. As a result, the compiler creates a file with extension `.class`. This file contains a Bytecode. We can execute this Bytecode on any operation system with a JRE. JRE stands for Java runtime environment. In the more detail, JRE contains Java virtual machine (JVM) and sets of a standard libraries. This JVM can execute the bytecode of the program. JVM can also execute the file with extension `.jar`, which contains `.class` files. JVM is sometimes called the interpreter. This process is shown in Figure 2.4.

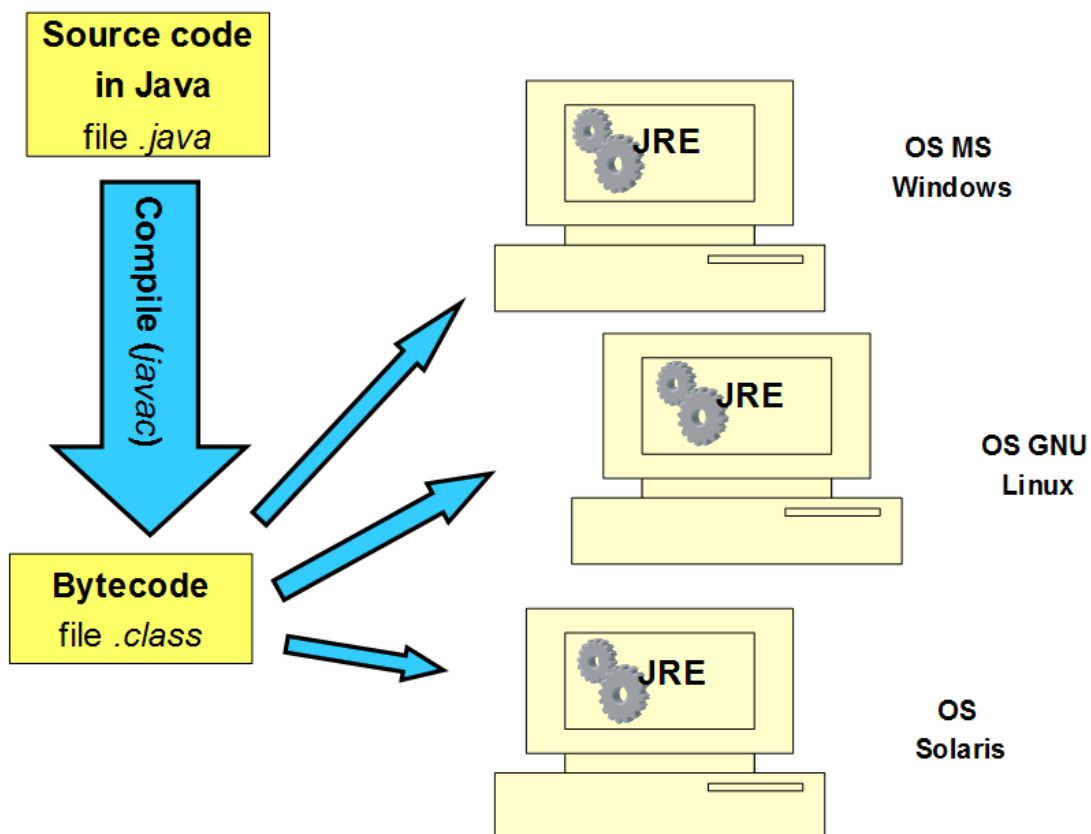


Figure 2.4. Creating and executing Java program

This is the different approach than software developing in C language, where the source code needs to be compiled in a specific platform's compiler (the win-

dows compiler, the ubuntu compiler...); hence, the C programs are not portable as Java programs. Beside the term JRE, there is a well known term JDK (Java Development Kit). JDK is a package that contains JRE, Java compiler, documentation and some supporting programs. To summarize, JRE is just used for executing programs but JDK serves for entire program development.

The basic steps for the Java applications development on Android are similar with a few exceptions. We also have to write the program in Java, compile the source code and run the program, but the main difference resides in the usage of SDK instead of JDK for the development. The bytecode generated with SDK is compiled to the Dalvik bytecode file with the extension .dex. This bytecode is different than the bytecode generated with JDK. Finally, this Dalvik bytecode is executed on the Dalvik virtual machine. The Dalvik virtual machine is also different from the Java virtual machine. In Figure 2.5, we can see the comparison between normal desktop java approach and Java android approach.

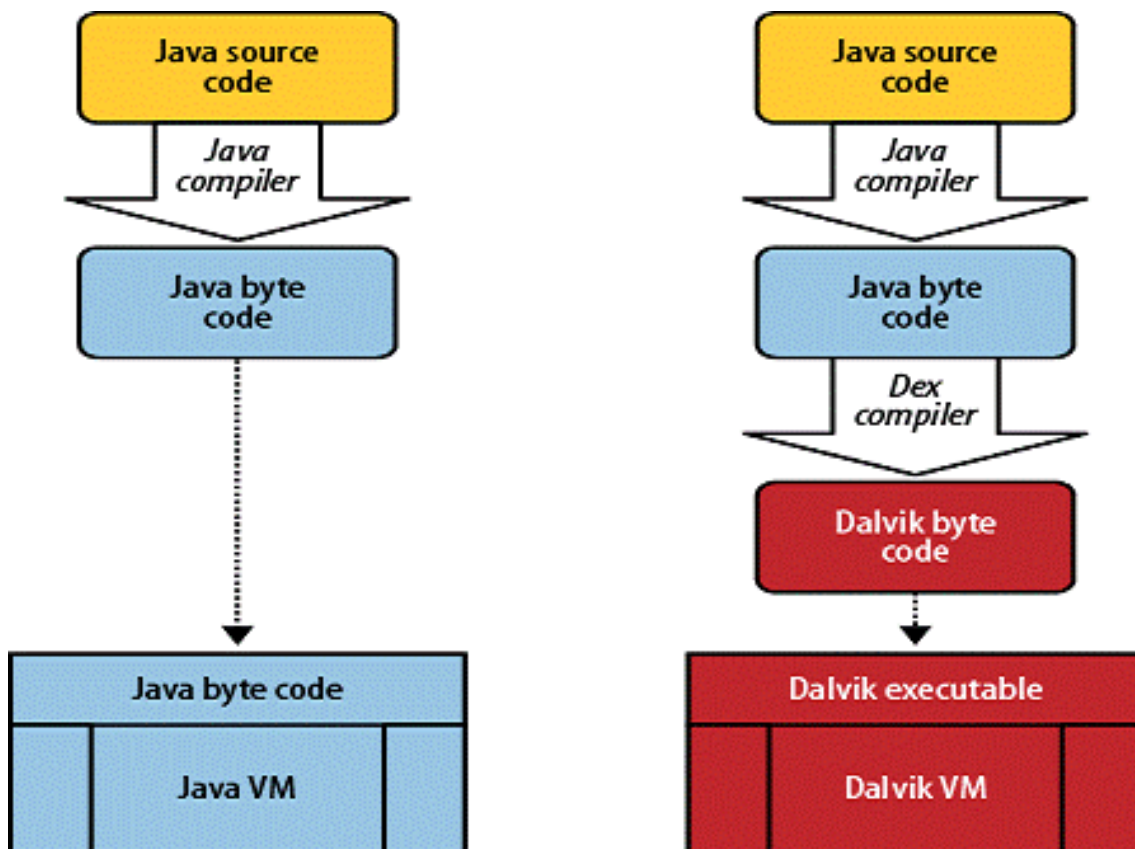


Figure 2.5. Creating and executing java program for the desktop and for the Android (JDK vs SDK, undertook from reference 8 in Appendix D)

In Figure 2.6, there is the structure of the Android platform. As you can see from Figure 2.6, the Android has the following layers:

- Applications
- Framework services and libraries
- Native libraries, daemons and services
- The Linux kernel (drivers for hardware, networking, file system access and inter-process-communication)

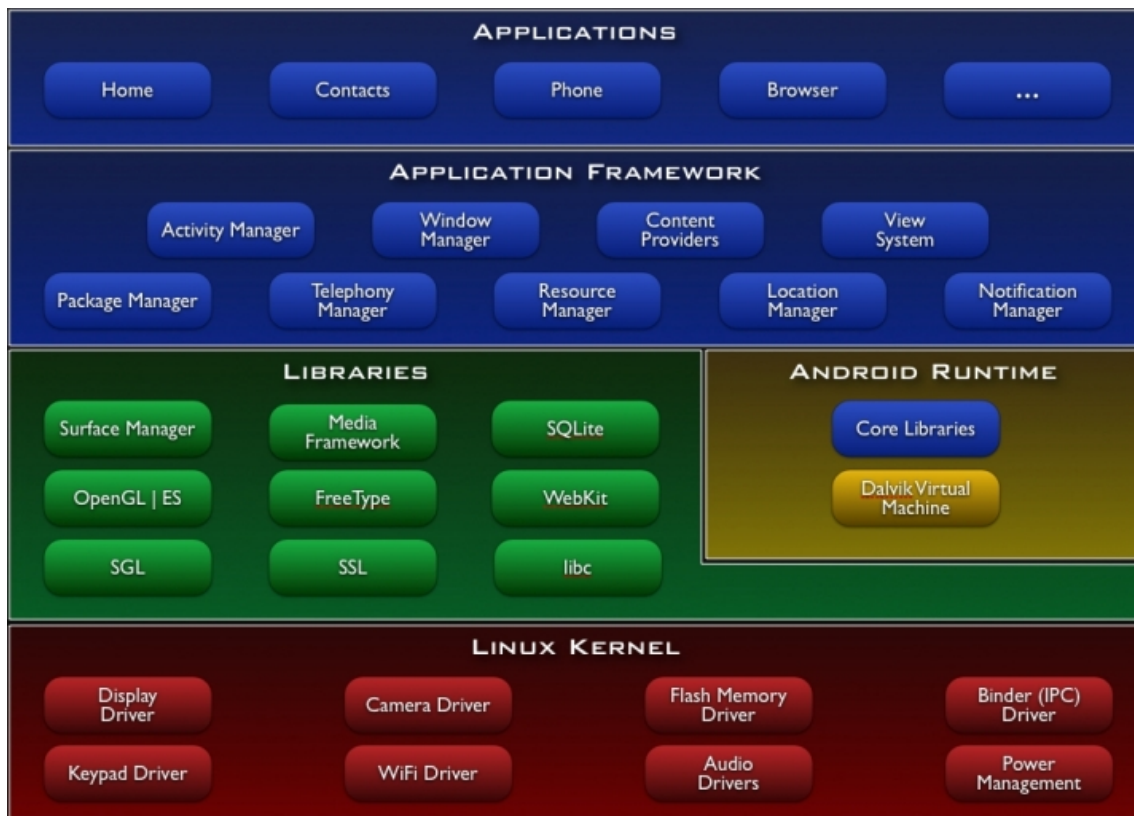


Figure 2.6. Structure of Android (undertook from reference 5 in Appendix D)

2.4 Life cycle of Android applications

The most important thing in android application is the Activity class. Each Activity instance in our application can change state in the Activity life-cycle. All of Activities are managed in activity stack. When new activity starts it is put on the top of the stack.

Every activity has 4 main states:

- Active or running
- Paused
- Stopped
- Finished or killed

The first state occurs when activity comes up in the foreground. If this activity lose the focus, the state changes to paused. It is still visible and alive. If our activity is totally covered by another activity, the state changes to stopped. Activity with this state is not visible. When activity has state paused or stopped the Android can remove the activity from memory. It can be done by asking it to finish or just kill its process. Figure 2.7 shows the diagram of the life cycle of the android activity. As we can see, there are 3 main lifetime loops.

- Entire lifetime
- Visible lifetime
- Foreground lifetime

The entire lifetime of the activity is called from the first call `onCreate(Bundle)` to the final call `onDestroy()`. The visible lifetime of the activity is called from the first call `onStart()` to the call `onStop()`. During this phase, the activity is shown on-screen. It does not have to interact with the user. The last mentioned lifetime loop of the activity is called the foreground lifetime. It is called from the first call `onResume()` to the call `onPause()`. During this time, the activity is on the top of the activity stack so it is in foreground and the user is interacting with this activity. Every activity needs to be declared in `AndroidManifest.xml`. In every activity, you will need to define at least `onCreate(Bundle)` where you are initializing your activity and connecting to your xml file with UI (or programmatically). You will very often need to implement `onPause()`, where you can handle the data storage etc., since the user can often leave your activity.

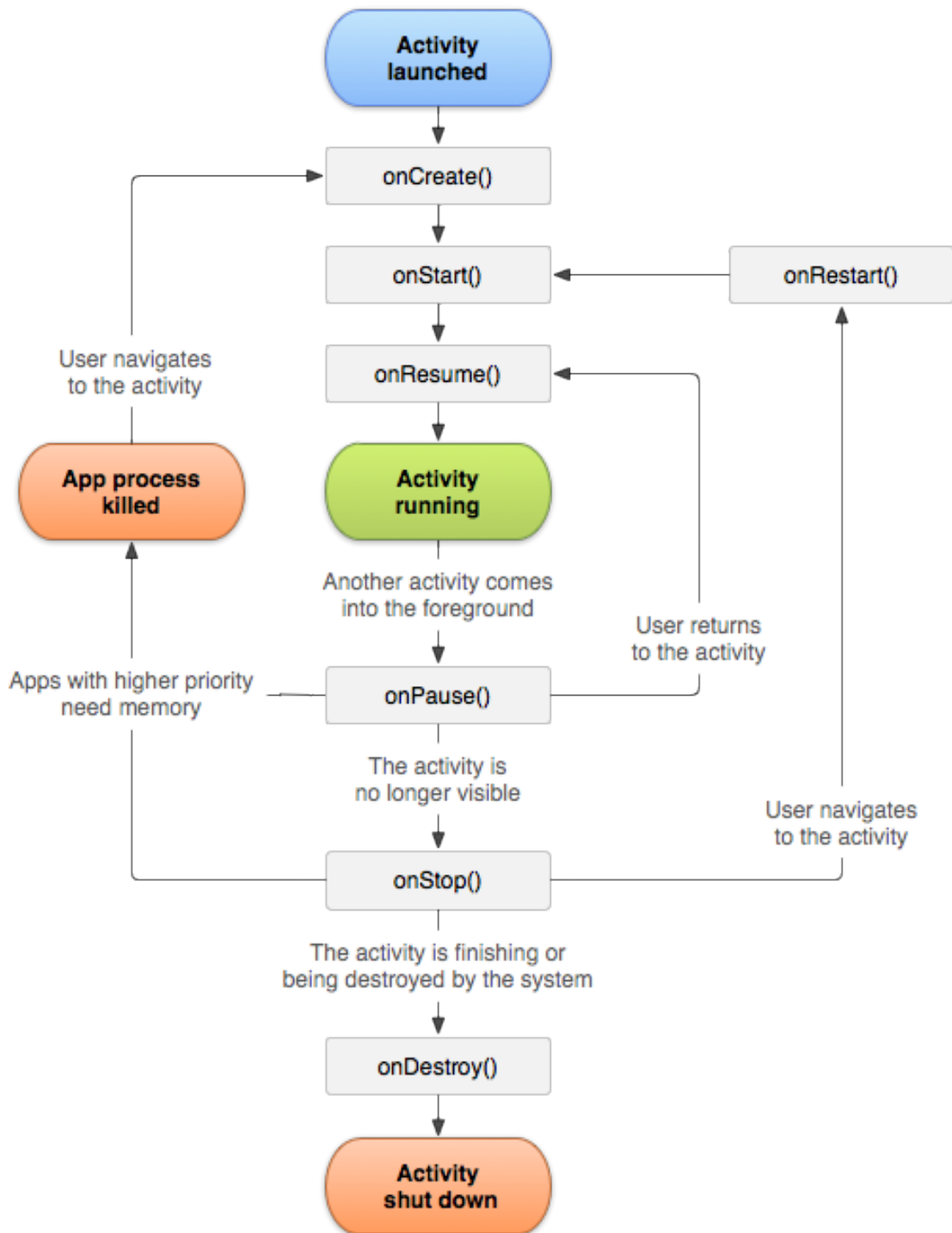


Figure 2.7. Life cycle of android activity (undertook from reference 5 in Appendix D)

Chapter 3

Related work

Approach in the articles [1,2] in Appendix D is not the only way how to generate UI from the model. The topic of User Interfaces generated from domain objects is mentioned in [9,10] in Appendix D. The framework is called Metawidget and it is based on Model driven development (MDD). The user just creates objects and puts them to Metawidget's framework. The UI is generated according to the model. Metawidget supports a lot of technologies from Android, Google Web Toolkit (GWT), HTML 5 (POH5), JavaScript to JSF and JSP. Metawidget works in three basic steps. First, Metawidget comes with a UI component native to your existing front-end. Second, Metawidget inspects, either statically or at runtime, your existing back-end architecture. Third, Metawidget creates native UI subcomponents matched to the back-end. In the articles [1,2] in Appendix D, the other aspects were added based on annotations. Metawidget adds these informations based on existing backend of any applications.

Model driven development (MDD) is based on the idea that the model should be primary centralized place for all informations. This models is then compiled or transformed in other way into the deployed application code. The benefits are reduction of informations in application and structure the informations into one place. The disadvantages can be adaptation and evolution management [11] in Appendix D. This approach does not go well with OOP, because we need to maintain the interconnection between the model with the backend of the application. There is another tool that we need to describe the additional informations. This tool in the MDD is called the Domain-specific language (DSL). Sometimes it is informally called mini-languages, because they describe the additional information in the inner of the other language. There are a wide variety of DSL. A domain-specific language can be one of a visual diagramming language, programmatic abstractions, declarative language (OCL) or even whole languages like XSLT. As we can see, some of them evolve into the programming tools that are used frequently (XSLT). Some of the domain-specific languages are not used now as often as they were (for example OCL).

Generative Programming (GP) is a specific type of a programming that generates the source code from domain-specific code. The goal is to improve productivity of developer, make way between application code and domain model, support reuse, adaptation, simplify management of components [12] in Appendix D.

Meta programming (MP) is a technique, which allows the developer to modify the structure and the behavior of the applications at the runtime. The reflection is one of the options how to implement the MP [13] in Appendix D. The developers can inspect the classes, the fields, the methods at the compile time and they do not even have to know their names at compile time. The MP allows the developer to adapt the application to the different situations. The bottleneck of this solution is the performance. The applications are significantly slower with the MP and are harder to test or debug then the applications without the MP. To deal with this problem the developer can use some cache.

Chapter 4

Framework requirements

A good aspect oriented framework for Android should support these attributes in order to be effective and desirable:

- multiple presentations
- multiple layouts
- security, visibility
- validity
- avoid mistakes and lower number of errors
- reduce duplication
- readable code fragments
- adapt to runtime application context
- be easy to develop and to maintain
- minimal amount of written code and efforts
- Android 2.2 and higher version
- Adroid SDK and plugin for Eclipse
- backend development

From the chapter 1, we know that we want to have implemented aspects like security, layout, input validation, data binding and presentation. We also want to support the developer's multiple choices in a part of the presentation (like integers, strings) and the layout (for example one column layout, two column layout). Very important part of requirements is to provide some security check if a logged user can see presented data or not. An another section is a validation. The normal validation is checking the validity of inserted data while we click on send button. The data are checked and based on the validation, the user is passed to another activity or the information about errors is provided. This is the basic validation. Another requirement on validation is to check default data that are preset to the instance before the form is even created. If this data are not valid, the user is informed. The last part of validation is checking validation of each element while the user is changing the value.

If we start using any framework, we are expecting clarity of its usage to avoid some mistakes and lower the number of errors. The reduction of duplication in

code is obvious. We do not want to write same or similar code always again and again; furthermore, we want to avoid the situations when the change of something in that redundant part of code is required which would result in modifications of all redundant parts of code. Instead of that, we want readable code fragments with no redundance. It will result in a nice reuseable code which will be separated according to each aspect and developer will exactly know where he needs to change something. This approach saves the time when an application is being developed and it is easy to maintain. Another requirement on this framework is that it has to be able to adapt to runtime application context. It does not even have to know your instances before the application is compiled to change layout etc., it will be created generic. One of the most important requirement is a minimal amount of written code and efforts. This point is joined up with the other points in the list like reduction of duplication, readable code fragments, be easy to develop and to maintain etc. (any framework is expected to accomplish these last mentioned requirements). There are lots of versions of Android OS and if you make some program for concrete version and its higher variants, it will not be working on lower Android versions. In Figure 4.1, there is a distribution of usage of the Android version used by users. As we can see versions under number 2.2 are not even mentioned on the list, because the number is too small. So the minimal SDK version 2.2 is a good choice. Of course, we need Android SDK and some plugin for IDE. In case of this framework, it is plugin for eclipse. There are also another plugins, for example plugin for Netbeans.

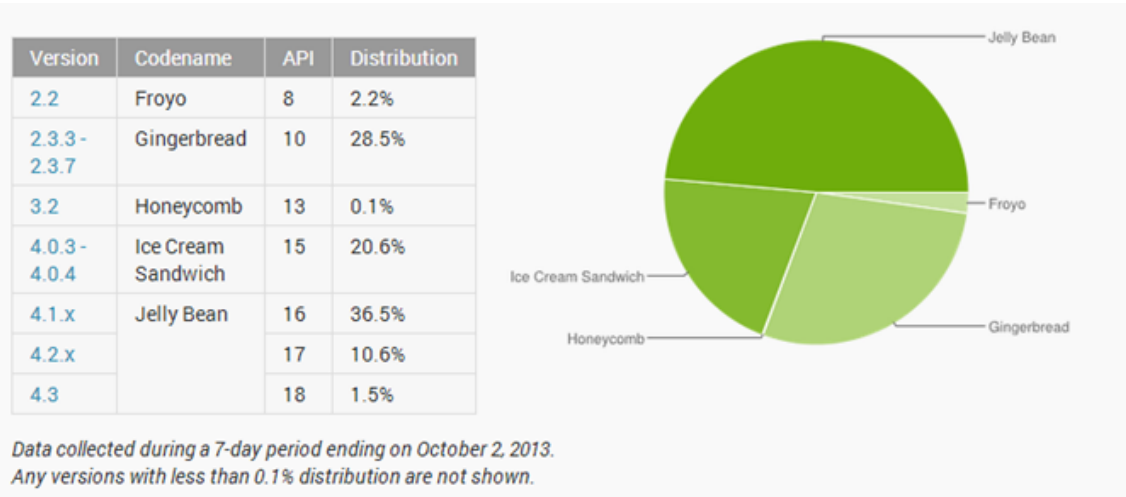


Figure 4.1. Distribution of Adroid versions (undertook from reference 7 in Appendix D)

We also do not want to only develop forms in our programs; hence, another requirement should be the backend development. The framework should allow us

not only to maintain our UI, but we should get space to develop some functionality in the backend of the application.

Chapter 5

Analysis and design of the framework

The whole framework is designed in a modular way so it is easy to extend it for example by another aspect that will come up in the future. In phase of designing, the important factor for the framework was platform on which the application will run. This platform is a mobile device.

5.1 Analytic model of classes

The analytic model of classes is a diagram which captures a general static view of the application. The purpose of this is to illustrate types of objects, variables and their relationships. Figure 5.1 shows this diagram. It does not contain all of the files (classes) because there would be much more objects and the diagram would not be easy to read, but it contains all packages and main functionality.

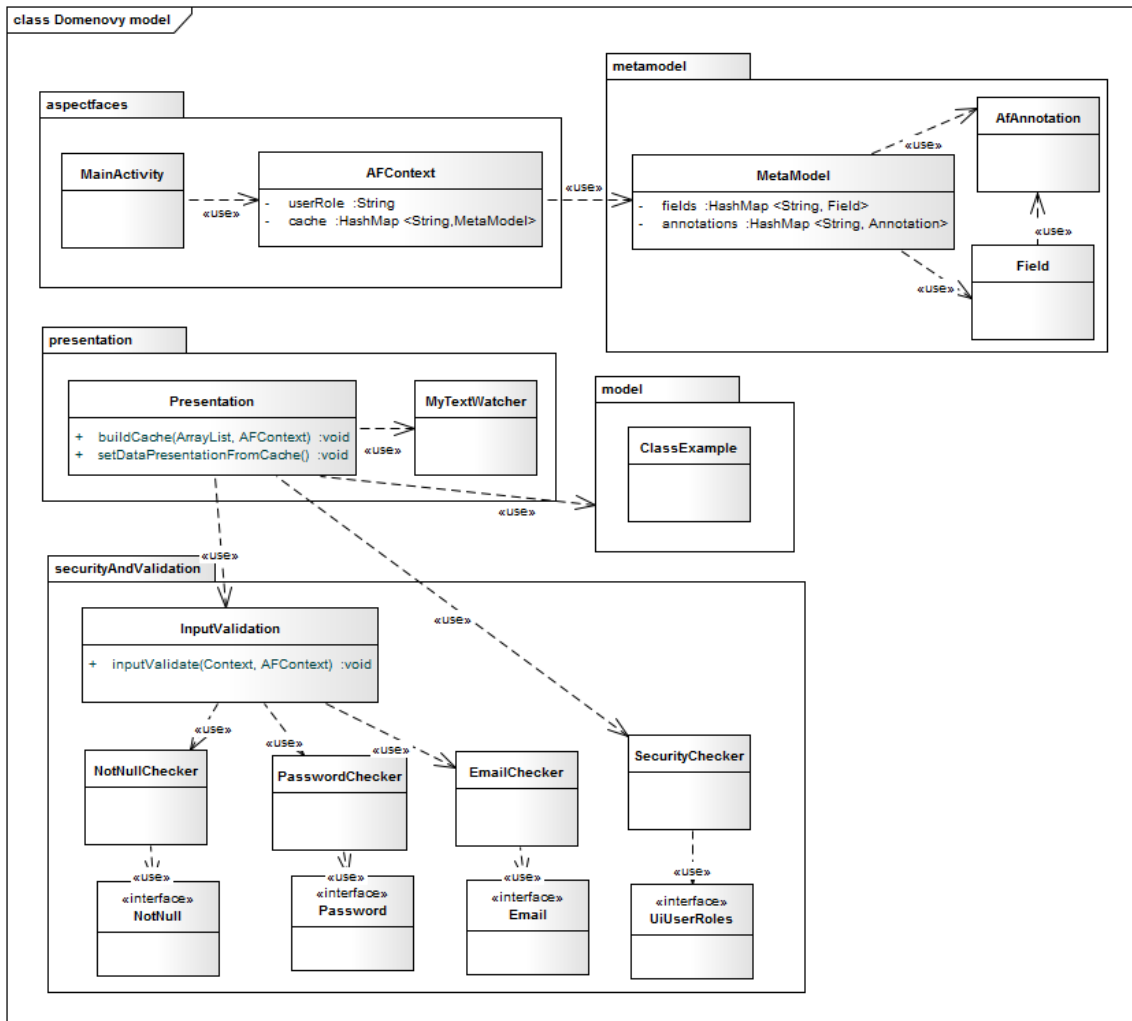


Figure 5.1. Analytic model of classes

5.2 Sequence diagram of frameworks lifetime

The sequence case diagram is used for a visualization of interactions between processes (objects). It also displays the right order of these interactions. It is a behavioral type of a diagram. Therefore, it is the best choice for showing how the framework works. The diagram includes parallel vertical lines called lifelines and horizontal arrows that represent the messages exchanged between them in the right order as they appear. On Figure 5.2, there is shown the sequence diagram. It shows what happens from the start of the application using the framework. The mandatory actions are create Presentation and call of functions buildCache(), setDataPresentationFromCache(). The other options of the framework are voluntary like on the diagram. If the developer wants to validate default data in created instances he just creates the InputValidation object and call inputValidate(). The framework will care about the rest via created rich entity (normal instance with

attributes and with annotations). In this diagram, it is also captured when the user changes the value of element it will call the listener which will call the inputValidation(). If the user sends the form of data it will also call inputValidation().

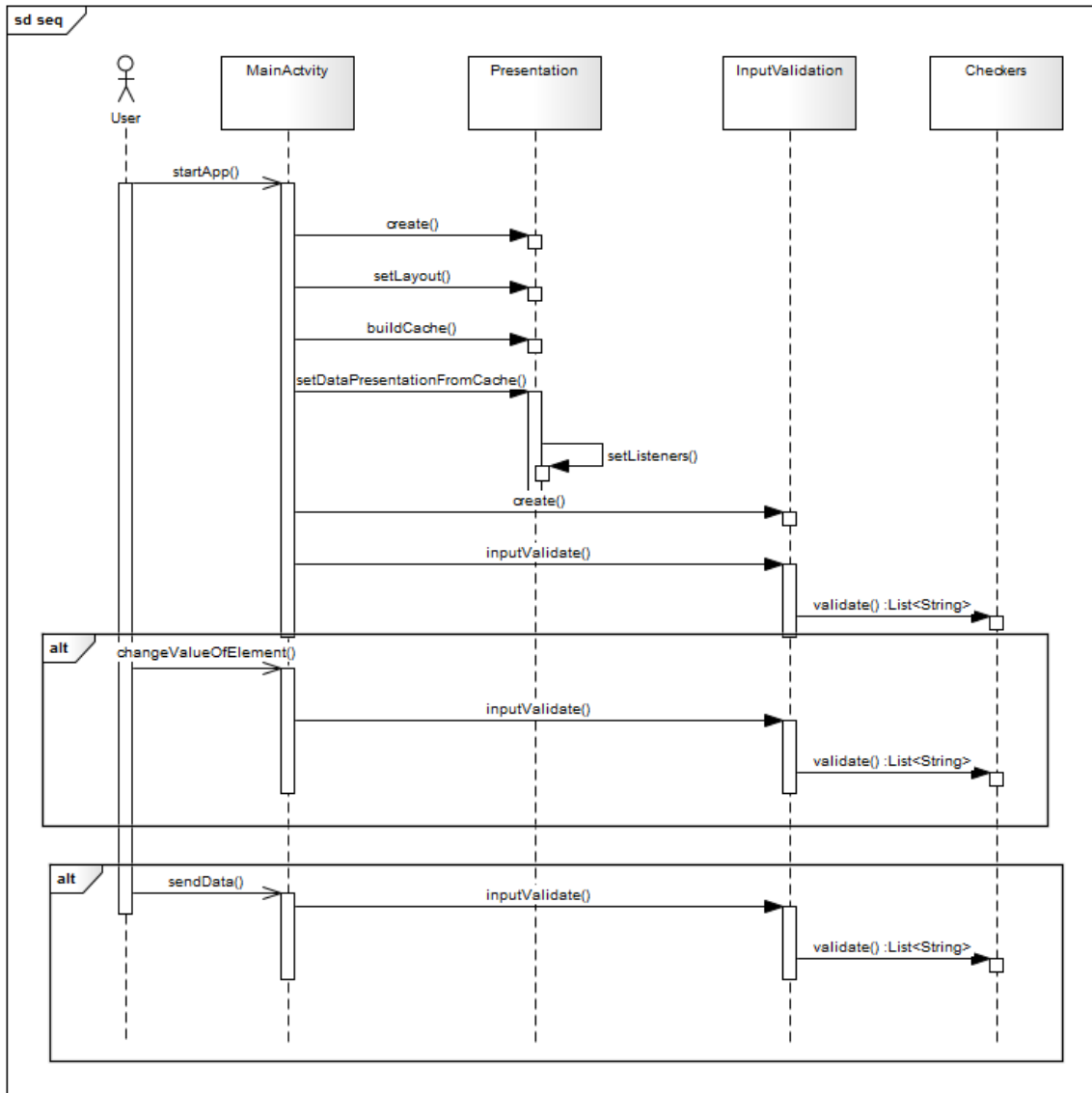


Figure 5.2. Sequence diagram of framework

Chapter 6

Implementation

As it was described in the chapter 2, the programming language Java without using any other external framework was chosen for the implementation. As a developing IDE was chosen Eclipse.

6.1 Directory structure

The directory structure of this framework is same as a normal Java Android application. As you can see from Figure 6.1 the project contains from folders, private libraries and other resources files.

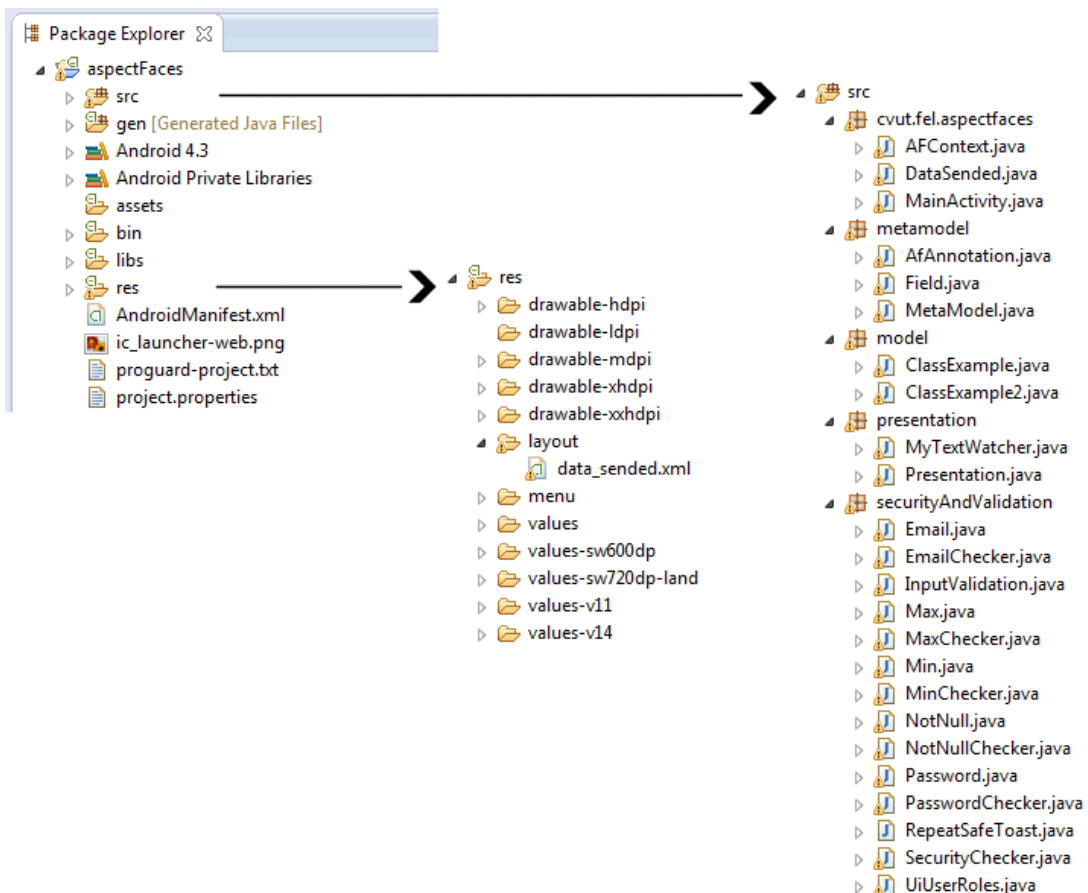


Figure 6.1. Directory structure of framework

The `src/` folder contains every java file in the project as it was said in the chapter 2.1. Java files are saved in the form of `src/package/javaFile.java`. The `gen/` folder contains java files generated by ADT. For example, the most important `R.java` file or `BuildConfig.java` file is there. The `R.java` file includes all the resources IDs provided to resources (drawables, layouts, styles etc). The `assets/` folder contains asset files which you may want to connect and use. In the `bin/`, there is the final apk file that is created when you compile the whole project. The private libraries are in the folder `libs/`. The resources take place in the `res/` folder. In Android, we have different types of resources. For every resource, we have a special folder with a special subfolder with xml files or even that resource like images. The type of resources are: color, drawable, layout, menu, values. All of them except `drawable/` folder contains xml files. In `drawable` folder, there are images used in the application. We can create every other resource in the specific xml file by writing xml tag with id and value. We can then use this resource many times in our application instead of just writing strings, menus and layout again and again. The example is shown in Figure 6.2.



Figure 6.2. Example how resources works

We can create any string ID we want and after compiling, SDK will create int ID for this name and it will add it into our `R.java` file. `@+` in ID means that it

is new ID and SDK will have to add it in R.java. As it was said in chapter 2.1, there is AndroidManifest.xml in which we have to register every activity or allow some permissions for the application.

The core of the framework structure is in the folder src/. It consists of 5 packages:

- cvut.fel.aspectfaces
- metamodel
- model
- presentation
- securityAndValidation

The first one cvut.fel.aspectfaces is mainly designed for the developer. Here, the developer can create any java file or folder of the project. MainActivity.java is the example file how developer should use this framework. AFContext.java is the frameworks context which includes some important information like a cache with metamodels, userRole and other. For every class that we will create, we need to create the metamodel in the cache. This metamodel class and its parts (fields and annotations) takes place in metamodel package. Model package is assigned for classes created by the developer. The Presentation part includes the data presentation, data binding and choosing layout. The SecurityAndValidation package is responsible for the input validation and security.

6.2 The cache of framework

The inner cache of the framework is the core part. In entity driven development, we regularly want to create the instance and create appropriate layout and other parts of the application. This approach is sufficient, but we want to use this approach for any instance we create so we need to do this more generic by using type Object and use Java reflection to get attributes and annotations. This way is better, but java reflection is not optimal for calling every time when we visit form. It would be too slow and if the application will be larger, it will become a bigger problem. The result of this is that we need some cache. Of course, we will create the cache with usage of Java reflection but only for the first time application is called. In this cache, we should save our metamodels effectively so the cache is the hashmap of metamodels. And every metamodel includes hashmaps of fields and annotations. At the end, every field has hashMap of annotations too. You can see this structure in Figure 5.1 in chapter 5.1. Hashmap is a Java hash based structure similar to the hashtable. It is used to store a key and value pairs. The

	Average	Worst case
Search	$O(1)$	$O(n)$
Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$

Table 6.1. Asymptotic time complexity of hashMap

advantage is asymptotic time complexity that is way much better then using Java reflection every time. Table 6.1 shows the main operations with time complexity.

As we can see from table 6.1, average time complexity is a constant for operations search, insert and delete. That first two operations are the most important for our purposes. The worst case can happen when collisions occure. In our cache, we are using the name of instance, field or annotation as a key and the value is the instance, field or annotation itself. The calling process is shown in Figure 6.3. The hashmap has one disadvantage that does not guarantee the order of elements stored but that does not pose the problem for our framework.

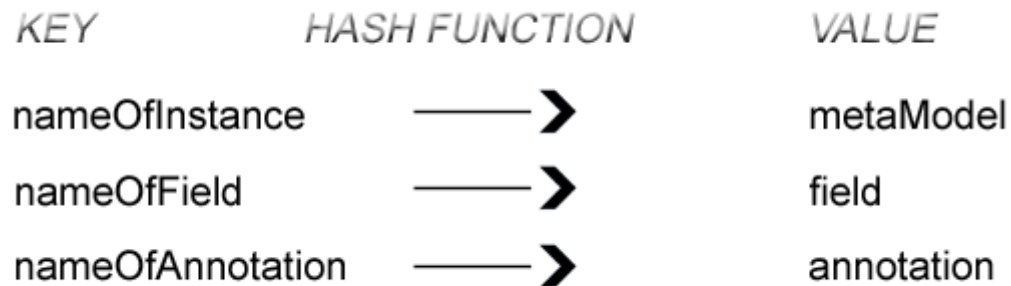


Figure 6.3. How hashMap works

In the practice, this will be used a lot because many android applications are based on fragments. The fragment is a dynamic UI and can be represented like subactivity. One activity can include many fragments and the user is switching between them. Sometimes it is called the multiple screen support or multi-pane UI. Because of that, the user is relaunching some fragments again and again. The time we save in recalling forms will pay off.

The cache is placed in AFContext. AFContext is a class which extends the Activity. This caused accessibility cache in whole application from any Activity. It is similar to some kind of a global class.

6.3 Data presentation

This is the part where we want to visualize the data into some elements. Each data can have different type so data are presented in each different way based on that type. Our framework has implemented 3 basic types:

- String
- int
- password

The last one password is based on annotation above attribute. This part is not only about show right editText (similar element as input tag in html) but also included the invocation of right software keyboard to put the right values. The framework also implements the @email annotation but it is primarily for value validation. The input values are same as normal String. In Listing 1 below you can see the declaration of attributes and the types.

```
@Password
String attr1;
int attr2;
String attr3;
```

Listing 1: Example use of data types

In Listing 2 below you can see the preparation of editText for String and in Listing 3 you can see the preparation of editText for int. The variable isPassword and isAllPassword are Boolean expressing if annotation @password was used. Variable isPassword is for annotation right above attribute and isAllPassword is for annotation above all instances. As you can see, this is not the preparation of layout, this is only the preparation of each element.

```
if(isPassword || isAllPassword){
    et.setInputType(InputType.TYPE_CLASS_TEXT |
        InputType.TYPE_TEXT_VARIATION_PASSWORD);
}else{
    et.setInputType(InputType.TYPE_CLASS_TEXT);
}
```

Listing 2: Preparation of editText for String

```
if(isPassword || isAllPassword){
    et.setInputType(InputType.TYPE_CLASS_TEXT |
    InputType.TYPE_CLASS_NUMBER);
    et.setTransformationMethod(PasswordTransformationMethod.getInstance());
}else{
    et.setInputType(InputType.TYPE_CLASS_TEXT |
    InputType.TYPE_CLASS_NUMBER);
}
```

Listing 3: Preparation of editText for int

You can see the result of rendered instance in Figure 6.4 where is shown the different types rendered. The different software keyboards that were invoked are shown in Figure 6.5.

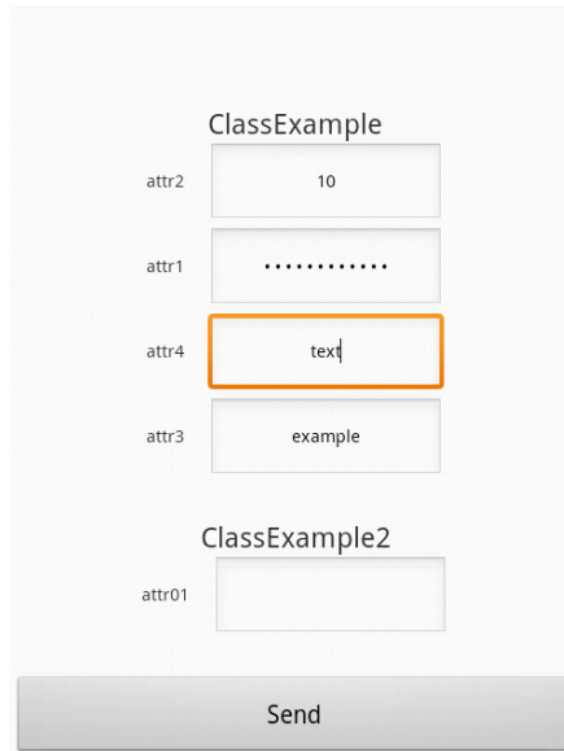


Figure 6.4. The result of presentation

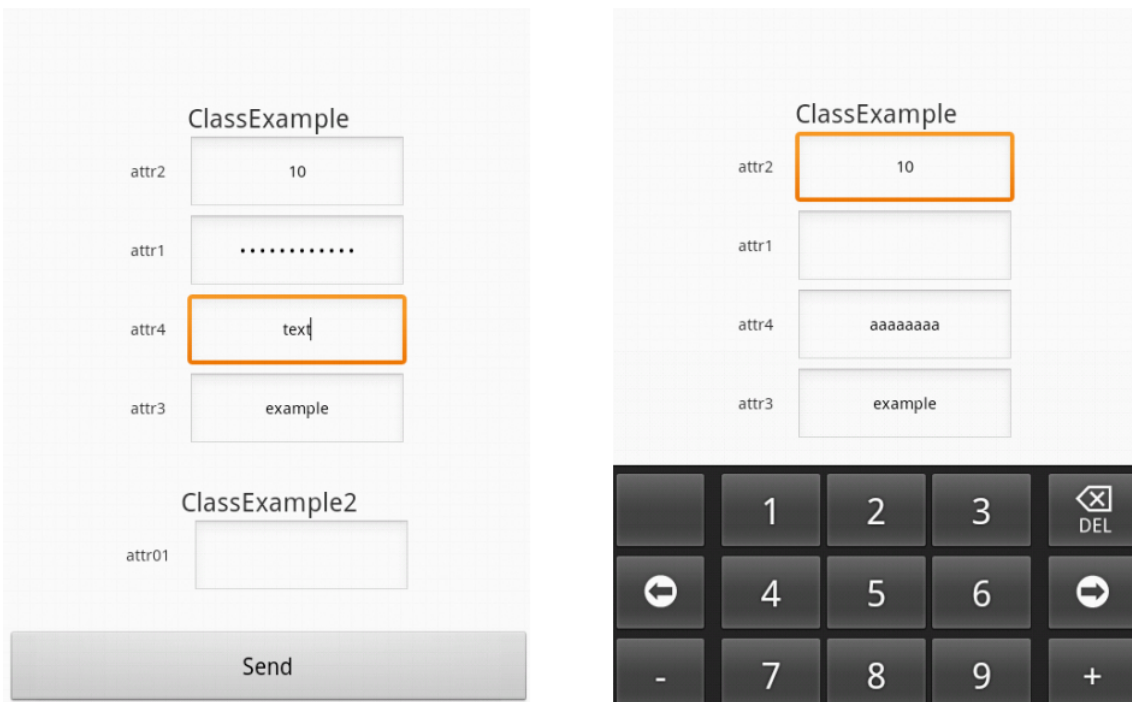


Figure 6.5. The result of presentation with invoked software keyboards

6.4 Data binding

There are two types of data binding: static and dynamic. The static data binding is term called when a type, name and value is resolved at the compile time. When it is resolved at the runtime, it is called dynamic binding. To resolve this at the runtime we can use java reflection. So we are using dynamic binding because we cannot resolve it at the compile time. If we look at the data from Listing 1 again we can set some default values to attributes like in Listing 4. It will simulate the default values that can developer create in the constructor like we did or it can be created dynamically in a program and then connected to these attributes.

```
this.attr1 = "";
this.attr2 = 10;
this.attr3 = "example";
this.attr4 = "aaaaaaaa";
```

Listing 4: Setting values

In this part we need also to ensure that the right informations are rendered. The header of one form is the name of instance. Then it is rendered a pair of textView and editText. TextView for name of attribute and editText for its value as you can see in Figure 6.4. The example contains of two objects. The first object has 4 attributes and the other 1 attribute with different types.

6.5 Input validation

The input validation is divided into three parts: checking default values, checking while changing values and checking before sending data.

Before the view is rendered the whole metamodel is checked if the values are correct. This is information for user for example if he has to fill some field if it is marked as @NotNull. In Figure 6.6, we can see information message about validation check. In the message, there is a list of all errors. This is optional part. If this validation check of default data is not required it is not called by developer. The Listing 5 illustrates the proper call of this validation type.

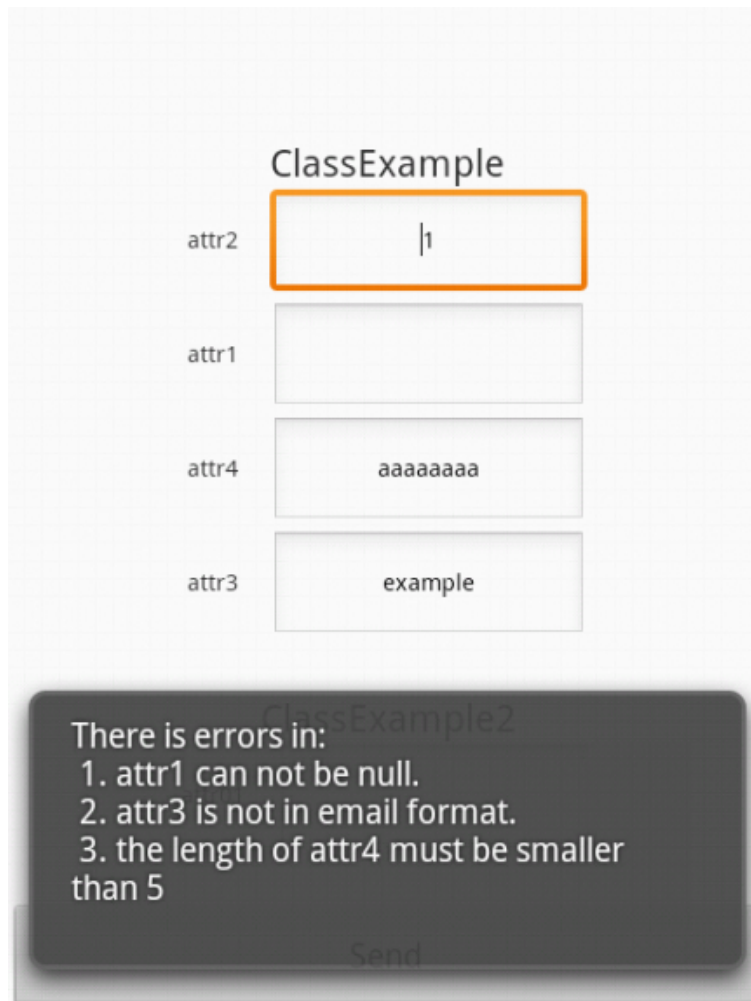


Figure 6.6. Input validation before view is rendered

```
InputValidation i = new InputValidation();  
i.inputValidate(this, afContext);
```

Listing 5: Optimal input validation

On all elements that user can change values are source of events for frameworks listeners. If the value is not right, user will be notified by toast message and the element will change its background color to red as you can see in Figure 6.7. The message contains information about errors of that one element. If the value will be changed to the right form it will change the background color back to the normal. For this event, the listeners and also class `MyTextWatcher` which extends `TextWatcher` are used. It is used to keep watch on the `editText` while the data is changed. `TextChangedListener` is used to detect changes and to attach the `MyTextWatcher` to the `editText`. The `TextWatcher` contains three main methods: `afterTextChanged`, `beforeTextChanged` and `onTextChanged`. The framework is

using `afterTextChanged` method and this method is responsible for invoking input validation. In Listing 6, we can see how `MyTextWatcher` is attached to the `editText` by `addTextChangedListener`.

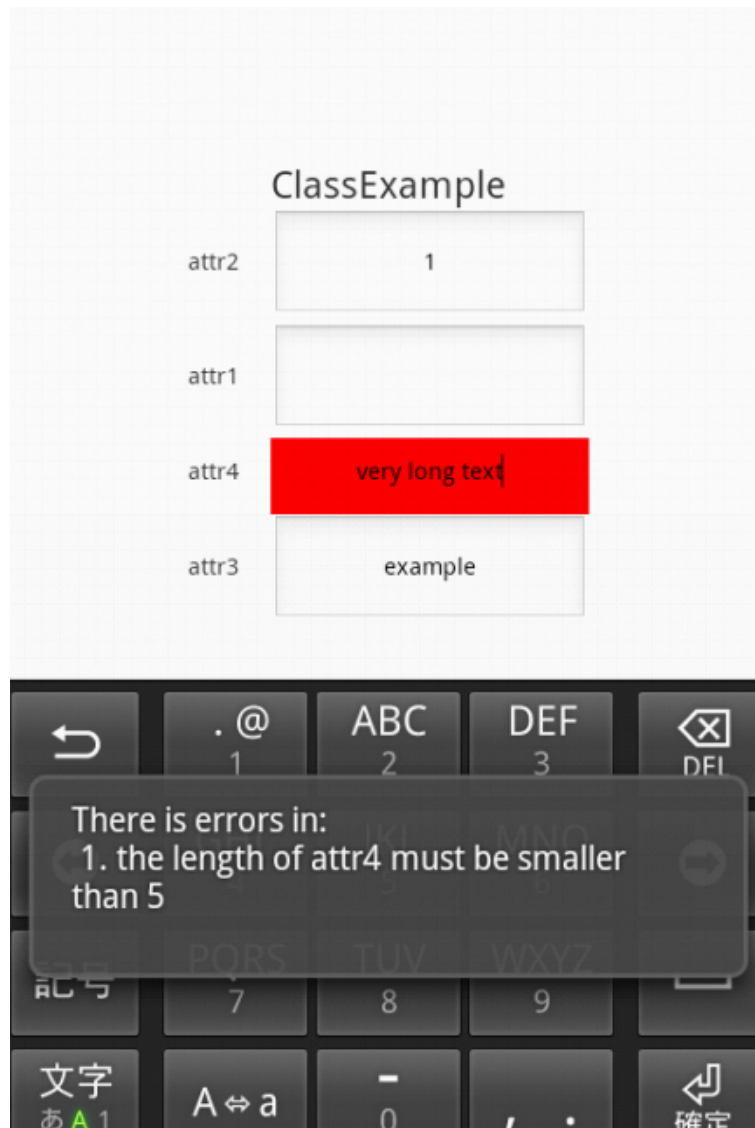


Figure 6.7. Input validation while the is changed

```
MyTextWatcher mtw = new MyTextWatcher(context, field, et);
et.addTextChangedListener(mtw);
```

Listing 6: `MyTextWatcher` attached to `editText` by `addTextChangedListener`

One of the biggest differences between this validation and the others is that this validation is checking just that one field and its value, not whole cache with all metamodels.

When the user wants to send data, there is the final input validation. Here, the framework checks whole cache whether every field is valid. If it is not, user will

be informed and it will not be allowed to another view. In the message, there is a list of all errors in all instances like the information message in the first part of the validation. If all fields are valid, it will allow user to another activity as you can see in Figure 6.8. This screen can be changed, it is up to the developer how it will look like. For example, it can lead to another form.

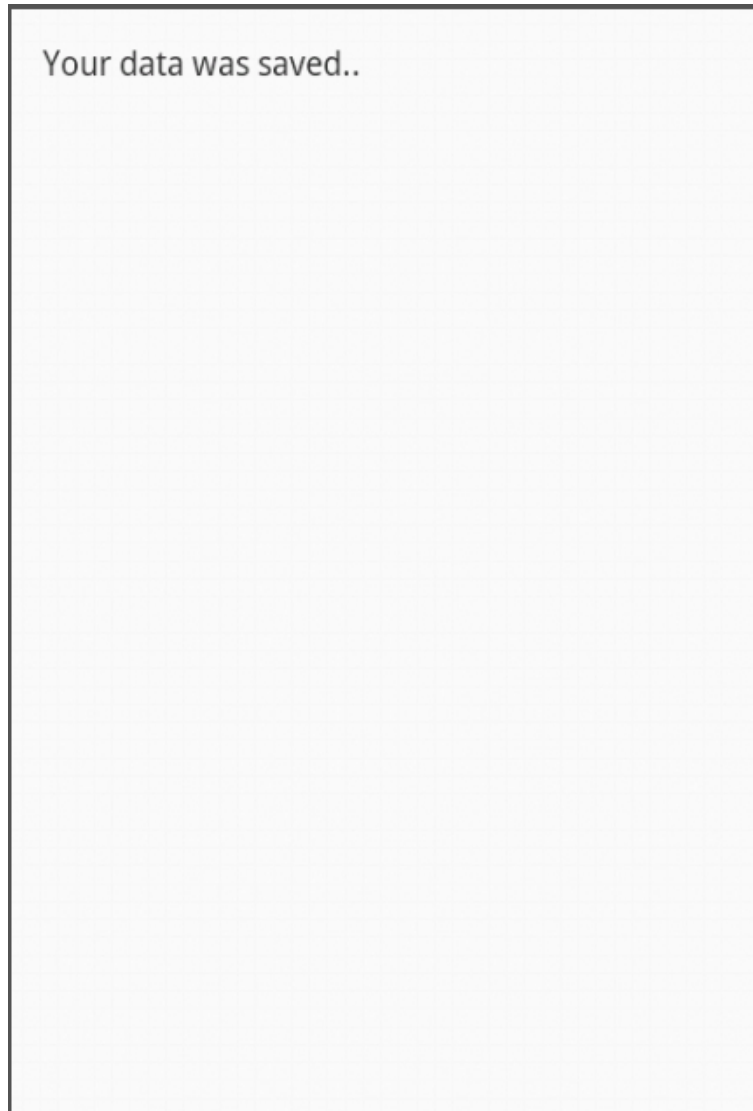


Figure 6.8. Data successfully passed validation

For support of this type of aspect, we will use annotations. The valid data can be restricted by these annotations:

- Email
- Min
- Max
- NotNull

In Listing 7, we can see the declaration of them with attributes.

```
@Min(size=1, message="the length of attr must be bigger than 1")
@Max(size=5, message="the length of attr must be smaller than 5")
>Email(message="attr is not in email format.")
>NotNull(message="attr can not be null.")
```

Listing 7: Annotations for validation

Every restriction is checked in `InputValidation` object. In detail, every restriction has its own checker that is returning list of error strings if there are any. These checkers are called from `InputValidation`. There is also difference between the annotations like `Email` and `NotNull` which has only one message that will appear to the user if the value is wrong. In these two annotations, we do not need any more information. In case of `Email` annotation, the checker is comparing string with regular expression. In case of `NotNull` annotation, the checker is comparing `String` with null or empty string. The annotations `Min` and `Max` has two attributes: `size` and `message`. The message will appear to user if the value is wrong in same way as for `Email` and `NotNull`. The `size` attribute is telling checkers of these two annotations what is the minimal or maximal boundary for values that are valid.

6.6 Layout

In this part, we want to let the developer to use some design layout. This is optional. If the developer will not set some layout, there is set a default one. The framework is supporting two layouts: 1 column layout and 2 column layout as you can see in Figure 6.9. The first one is default.

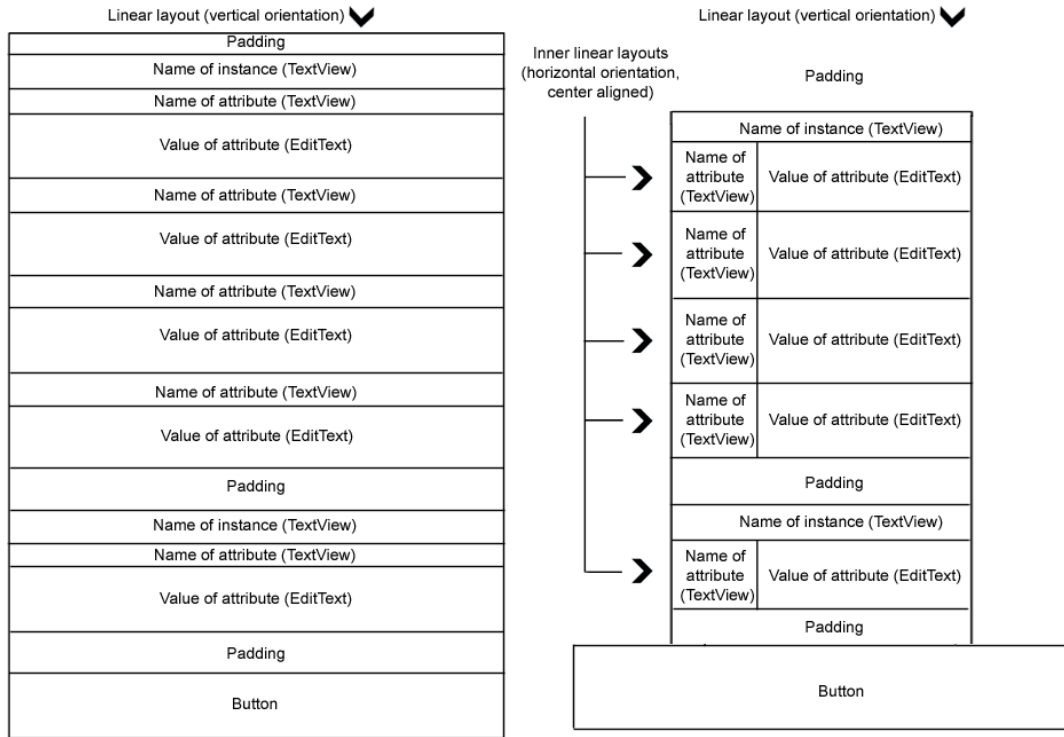


Figure 6.9. Templates of basic two layouts

In Listing 8, you can see how to set up the second layout. The first layout is just one `LinearLayout` with vertical orientation. All the elements are inside one by one. The second layout has outer `LinearLayout` with vertical orientation. But every row for any attribute contains inner `LinearLayout` with horizontal orientation. This inner layout contain `textView` for the name and `editText` for the value inside.

```
p.setLayout("linear2");
```

Listing 8: How to set up another layout

In Figure 6.10 you can see the result of both layouts.

Figure 6.10. The real view of basic two layouts

6.7 Security

Another aspect is the security. As a developer, we sometimes do not want the every field to be visible to everyone. For certain users, we can create a special form. Generally, we have to divide users to roles. In whole application, the developer can implement login or register as he wish but he has to store String with role into AFContext to variable userRole. The framework then supports this aspect through the annotation `@UiUserRoles`. This annotation has one attribute named role. It is an array of Strings that represents roles which are allowed to this attribute. If the attribute does not have this annotation, it is visible for all users. If the instance or attribute has this annotation, it is visible only to the user with one of roles from this array. The example of usage you can see in Figure 6.11. The instances were created from classes in Listing 9.

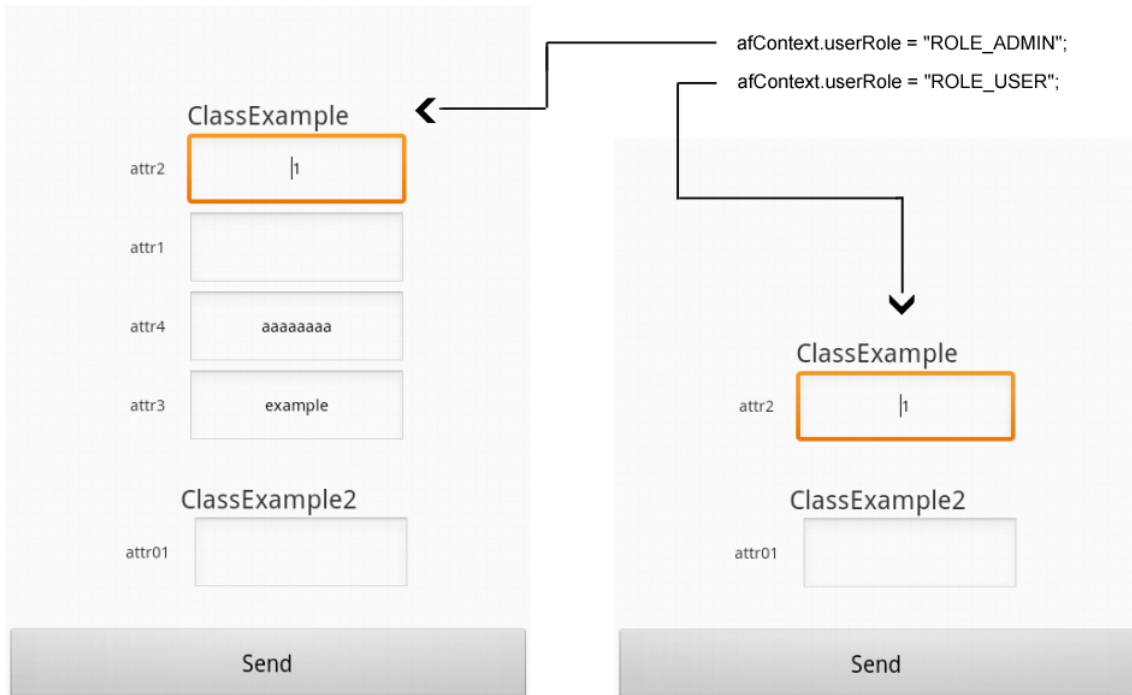


Figure 6.11. The example of usage security

```

@UiUserRoles(role={"ROLE_ADMIN"})
public class ClassExample{
    @NotNull(message="attr1 can not be null.")
    @Password
    String attr1;
    @UiUserRoles(role={"ROLE_ADMIN","ROLE_USER"})
    @NotNull(message="attr2 can not be null.")
    int attr2;
    @UiUserRoles(role={"ROLE_ADMIN"})
    @Email(message="attr3 is not in email format.")
    String attr3;
    @Min(size=1, message="the length of attr4 must be bigger than 1")
    @Max(size=5, message="the length of attr4 must be smaller than 5")
    String attr4;
}

public class ClassExample2{
    @UiUserRoles(role={"ROLE_ADMIN","ROLE_USER"})
    String attr01;
}

```

Listing 9: Setup of instances with security

6.8 Error messages

There are several ways how to inform the user about errors in values that has been filled: list of errors at the bottom or top of form, label next to invalid value

or invoke toast with list of errors. The first option is good in the case of desktop application. For the mobile application this list occupies a lot of space and this is not good for mobile application. When the form is even larger the list can be also long and it will slower the user and the view will not be clear. Other disadvantage is that every error is not pair directly to the element which is wrong filled. The second option is better than the first one because it is paired with error element but it has also the same problem. It occupies a lot of space next to element. In the worst scenario, it will double the space of form. I have chosen the third option for framework. The reason is that toast messages are naturally used in android applications. The toast message contains a list of errors like first option but it is above the entire form so it does not occupy another space on screen. Also the toast do not lock the screen and user can still type into elements. This option still have one disadvantage. The toast is temporarily displayed and then it disappear. All the other options are permanent and user can see it all the time. The information about wrong input is highlighted there by red background so it may not make troubles to users.

These toast messages have however one problem. If the user types a long word or just click on button send it invokes a lot of toast messages that occure right after each other. In the framework, there is implemented own class RepeatSafeToast. Every time this class is called to show toast, it checks few conditions. The first condition is that the message of toast has changed. And the second condition is that the time of last toast plus duration of toast has expired. When this check is passed (class cannot find the pair, the key and the value for this toast in hashmap), the record is stored in hashMap and new toast message is invoked. Hashmap contains pairs of key and value. As a key, it is used message of toast and as a value, it is used time. In Listing 10, we can see the usage.

```
RepeatSafeToast.show(myContext, (String)text);
```

Listing 10: The usage of RepeatSafeToast

Another important setup of toast is lenght of the duration. There is two default values:

- SHORT_DELAY
- LONG_DELAY

SHORT_DELAY is 2 seconds and LONG_DELAY is 3,5 seconds. I picked LONG_DELAY for better vision of the information to user. With the

SHORT_DELAY could easily miss the error message. And if even this happened with this setup it is not problem for him to invoke it again (by clicking send button or change value in editText).

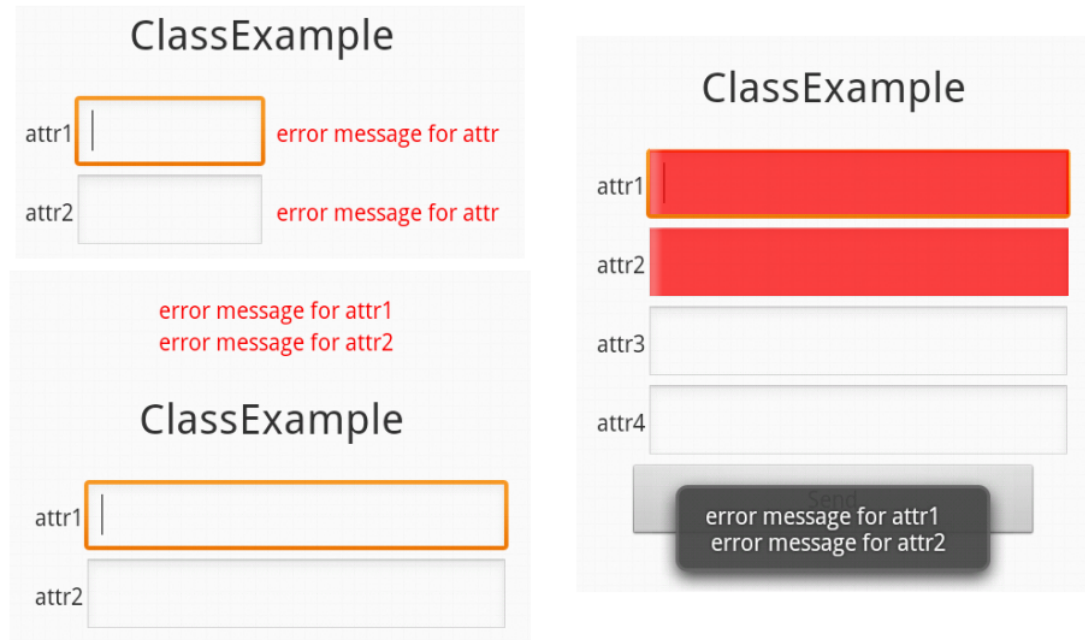


Figure 6.12. The options of error messages

In Figure 6.12 you can see all three options. On the left side on the top there is error message next to every element. As we can see editText has to be smaller cause of size of screen. Under this example is option where the list of errors is above the form. The space is also enlarged. On the right side there is option with the toast. As we can see no more additional space is required.

Chapter 7

Comparison of aspect oriented approach and conventional approach for Android platform

In the chapter 6 Implementation, there was described the aspect oriented approach. In the conventional approach, the presentation layer was implemented in xml, the data binding in Java, input validation in Java, layout in xml and security in Java. The project was developed to implement one form that is the same as the first one in the example of aspect oriented approach with four attributes. All aspects was coded in common way and not stored in annotations like in the aspect oriented approach. If we want to do another form, we will have to write the similar amount of code. This code is redundant and is making application hard to maintain. The big difference against AOP is in security, where in aspect oriented approach we just do not create particular element. Here, we create all elements and then we are changing visibility if the user has particular user role.

The advantages of the aspect oriented approach are shown in table 7.1. Here, we can see that AOP is definitely better in reuse, reduction of the code, maintenance the code, separation of aspects and readability of code. The reuse in conventional approach means copy, paste and edit. That is not good approach. From table 7.1, we can also see the difference in lines of code (LOC). The AOP has 4 lines of Java code and 25 of Java class. The conventional approach has 16 lines of Java class, 377 of Java code and 102 of xml. LOC where counted by regular expression in search function in eclipse IDE. LOC is counted from the view of developer so the body of the framework is not counted. As we know, this is example that has only 4 attributes. If this number rises for example to eight the LOC for conventional approach will also rise by similar amount lines. On the other hand, AOP will increase only by about ten LOC (four lines plus some annotation).

As we can see, the standard deviation for AOP is smaller. As a result, we can state that these two distributions are probably different.

$$s_x^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

Figure 7.1. Formula of standard deviation

In Listing 11, there is the example of how the time was taken. The start time is taken before all the code in both approaches and the end time is taken on the bottom. The result is difference between these two values.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //begin time
    long lStartTime = new Date().getTime();

    /*
    code...
    */

    //end time
    long lEndTime = new Date().getTime();
    long difference = lEndTime - lStartTime;
    System.out.println("Time to launch AOP app in milliseconds: "
        + difference);
}
```

Listing 11: Usage how to get time of launching main activity

You can see the three examples of each prints of the launching time in Figure 7.2. The top three prints are from the conventional application and the bottom three are from the AOP application.

7. Comparison of aspect oriented approach and conventional approach for Android platform

Search for messages. Accepts Java regexes. Prefix with pid, app, tag; or text: to limit scope. verbose

L...	Time	PID	TID	Application	Tag	Text
D	04-27 18:05:0...	32	32		dalvikvm	paused 155ms GC_EXPLICIT freed <1K, 53% free 2538K/5379K, external 1625K/2137K, paused 449ms
I	04-27 18:05:0...	450	450	com.examp...	System.out	Time to launch conventional app in milliseconds: 210
I	04-27 18:05:0...	60	97	system_pr...	Activit...	Displayed com.example.conventionalapproach/.MainActivity: +1s209ms
D	04-27 18:05:0...	139	139	com.andro...	dalvikvm	GC_EXPLICIT freed 103K, 51% free 2889K/5895K, external 4776K/5279K, paused 72ms
I	04-27 18:01:2...	417	417	com.examp...	System.out	Time to launch conventional app in milliseconds: 171
I	04-27 18:01:2...	60	97	system_pr...	Activit...	Displayed com.example.conventionalapproach/.MainActivity: +1s179ms
D	04-27 18:02:1...	60	126	system_pr...	SntpClient	request time failed: java.net.SocketException: Address family not s
I	04-27 18:00:4...	384	384	com.examp...	System.out	Time to launch conventional app in milliseconds: 196
I	04-27 18:00:4...	60	97	system_pr...	Activit...	Displayed com.example.conventionalapproach/.MainActivity: +1s180ms
D	04-27 18:00:4...	139	139	com.andro...	dalvikvm	GC_EXPLICIT freed 108K, 50% free 2957K/5895K, external 4776K/5279K, paused 72ms

Search for messages. Accepts Java regexes. Prefix with pid, app, tag; or text: to limit scope. verbose

L...	Time	PID	TID	Application	Tag	Text
I	04-27 18:08:2...	516	516	cvut.fel....	System.out	2. attr3 is not in email format.
I	04-27 18:08:2...	516	516	cvut.fel....	System.out	3. the length of attr4 must be smaller than 5
I	04-27 18:08:2...	516	516	cvut.fel....	System.out	Time to launch AOP app in milliseconds: 114
I	04-27 18:08:2...	516	516	cvut.fel....	System.out	end AF
I	04-27 18:08:2...	60	97	system_pr...	Activit...	Displayed cvut.fel.aspectfaces/.MainActivity: +897ms
W	04-27 18:08:2...	60	158	system_pr...	InputMa...	Starting input on non-focused client com.android.internal.view.IInp utMethodClient\$Stub\$Proxy@406aa2a8 (uid=10038 pid=450)
I	04-27 18:06:1...	483	483	cvut.fel....	System.out	Time to launch AOP app in milliseconds: 128
I	04-27 18:06:1...	483	483	cvut.fel....	System.out	end AF
I	04-27 18:06:1...	60	97	system_pr...	Activit...	Displayed cvut.fel.aspectfaces/.MainActivity: +848ms
W	04-27 18:06:1...	60	66	system_pr...	InputMa...	Starting input on non-focused client com.android.internal.view.IInp utMethodClient\$Stub\$Proxy@406aa2a8 (uid=10038 pid=450)
I	04-27 18:10:2...	549	549	cvut.fel....	System.out	Time to launch AOP app in milliseconds: 121
I	04-27 18:10:2...	549	549	cvut.fel....	System.out	end AF
I	04-27 18:10:2...	60	97	system_pr...	Activit...	Displayed cvut.fel.aspectfaces/.MainActivity: +889ms

Figure 7.2. The launching times of application

Chapter 8

Comparison of aspect oriented programming (AOP) for Android platform and Java EE

In this chapter, we are comparing AOP for Android platform and Java EE. The first big difference is that each of frameworks are used to create a little different applications. The first category contains applications only for mobile devices and the second is basically for enterprise web applications. The android applications are usually smaller then large enterprise application. As a developer for mobile devices, you need to focus a lot on a view because space is much more important then on desktop or web applications because you just have limited space for the view. Usability is also important. Every UI on mobile device has to be simple and easy to use. Java EE applications are huge enterprise applications deployed on web servers. They can be used on any device with connection to the internet. But primarily, the huge enterprise applications are focused on the desktop computer because on mobile device it is slower to control some use cases than on the normal desktop computer or notebook. For example, web applications can also be run on browser on IOS and not only on Android as the native application.

The aspect oriented framework, called Aspect Faces for Java EE, was created on a similar idea as for Java EE, but they are not implemented in a same way. The framework for Java EE is called by the tag in a view part as we can see the usage in Listing 12. Most of the times it is placed in JSP page or some xhtml page. In Listing 12, there is an example how to create two forms.

```
<!-- Form1 generated via Aspect Faces -->
<af:ui instance="#{bean.entity1}" edit="true"/>
<!-- Form2 generated via Aspect Faces -->
<af:ui instance="#{bean.entity2}" edit="true"/>
```

Listing 12: Usage of Aspect Faces in Java EE

Instead of this approach, the framework for Android is called in java activity class that user creates. The reason for this is simple. In Android application

Chapter 9

Table of regressive tests

Tested function	Input	Result
Set layout with method <code>setLayout()</code> of instance <code>Presentation</code>	String of layout name	Layout of form was changed
Create cache with method <code>buildCache()</code> of instance <code>Presentation</code>	None	Cache was created
Set presentation with method <code>setDataPresentationFromCache()</code> of instance <code>Presentation</code>	None	Return the final view of screen
Validate input values of elements with method <code>inputValidate()</code> of instance <code>InputValidation</code>	Context of android application (actual java activity), the frameworks global class <code>afContext</code>	Notify user about all errors in input values
Send valid data	Valid data	User will be forwarded to send activity
Send not valid data	Invalid data	User will be notify about error
Change user role and set security annotations	User role	Framework will create view according to user role
Change activity to forward user after sending data	Activity class name	User will be forwarded to new activity
Add annotation <code>@password</code>	None	The view element has changed with password view
Check validation of data that is inserted into field of form	Input data (String, int...)	The data is checked when user changed data in field

Table 9.1. Regressive tests

Chapter 10

Installation

10.1 The pick of operation system (OS) and IDE

It is recommended to use the framework under Windows 7 and using ADT in eclipse IDE. For this combination, the framework was tested. On the other OS Windows 8, Ubuntu and other linux OS, it should also work but it is not tested.

10.2 Instalation steps

- Insert instalation CD do mechanics.
- Extract AspectFaces.zip and his content store on disc.
- Open eclipse.exe and import framework as project or if you do not want to work in eclipse or you already have a project you are working on, use your IDE and copy packages in src folder to your project.
- Done.

Chapter 11

Conclusion

The diploma thesis was created according to the assigned task under the Department of Computer Science and Engineering, the Faculty of Electrical Engineering, the Czech Technical University in Prague within the study program Electrical Engineering and Computer Science, the branch Information Technology.

The task of the diploma thesis was to study the existing solutions for the aspect oriented user interface development for the platform Java EE from reference [1,2,3] in Appendix D, which provided the theoretical background, and to design the similar solution / framework for creating mobile applications for the Android.

The Android framework was created with the special consideration of the following aspects: security, layout, input validation, data binding and presentation. The application generated by this framework was compared with the application created with the use of the conventional approach. The application generated by the framework was better then the application created in conventional way in every feature: Reuse, Runtime approach, Reduce code, Better to maintain, Separated each aspects, Readable code, Time to launch the form (average) and Lines of code. The diploma thesis also contains the comparison of AOP framework for Android platform and for Java EE. The frameworks for both platforms are based on similar ideas. They implements same aspects in different way, but the result is essentially the same. The main difference is in targeted platforms.

All requirements from chapter 4 are integrated in framework. Every functionality in this work was tested (introduced in chapter 9). Accomplished tests affirmed full functionality of framework; therefore, we believe that all assigned tasks were succesfully solved. The proposed framework is available on the CD, which is attached to this diploma thesis.

Appendix A

Specification

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Jiří Šebek**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Výpočetní technika

Název tématu: **Aspektově orientovaný vývoj uživatelských rozhraní
pro Android aplikace**

Pokyny pro vypracování:

Seznamte se s existujícím řešením pro aspektově orientovaný vývoj uživatelského rozhraní pro platformu Java EE [1,2,3]. Navrhněte podobné řešení / framework pro vytváření mobilních aplikací na platformě android. Při zpracování tohoto řešení se soustředte na tyto dílčí aspekty: bezpečnost, rozvržení prvků (layout), validace vstupu, provázání dat, prezentace. Demonstrujte řešení na příkladech a testovací aplikaci, zhodnoťte výhody či omezení daného řešení. Porovnejte rozdíly mezi řešením pro platformu Android a Java EE.

Seznam odborné literatury:

[1] Tomas Cerny, Michael J. Donahoo, and Eunjee Song. 2013. Towards effective adaptive user interfaces design. In Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS '13). ACM, New York, NY, USA, 373-380. DOI=10.1145/2513228.2513278 <http://doi.acm.org/10.1145/2513228.2513278>

[2] Tomas Cerny, Karel Cemus, Michael J. Donahoo, and Eunjee Song. 2013. Aspect-driven, Data-reflective and Context-aware User Interfaces Design. In Applied Computing Review, Vol. 13, Issue 4, ACM, New York, NY, USA, 53-65. ISSN 1559-6915 <http://www.sigapp.org/acr/Issues/V13.4/ACR-13-4-2013.pdf>

[3] Tomas Cerny and Eunjee Song. 2011. UML-based enhanced rich form generation. In Proceedings of the 2011 ACM Symposium on Research in Applied Computation (RACS '11). ACM, New York, NY, USA, 192-199. DOI=10.1145/2103380.2103420 <http://doi.acm.org/10.1145/2103380.2103420>

Vedoucí: Ing. Tomáš Černý, MSc.

Platnost zadání: do konce letního semestru 2014/2015


doc. Ing. Filip Železný, Ph.D.
vedoucí katedry




prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 2. 2014

Appendix B

Symbols

AOP	Aspect oriented programming
AF	Aspect Faces
OS	Operation systém
UI	User interface
IDE	Integrated development environment
Java EE	Java Enterprise Edition
SDK	Software development kit
XML	Extensible Markup Language
CD	Compact disc
OOP	Object oriented programming
READ	Rich Entity Aspect/Audit Design
JVM	Java virtual machine
JRE	Java runtime enviroment
JDK	Java development kit
ADT	Android development tools
LOC	Lines of code
ms	millisecond

Appendix C

Code examples

- Listing 1: Example use of data types
- Listing 2: Preparation of editText for String
- Listing 3: Preparation of editText for int
- Listing 4: Setting values
- Listing 5: Optimal input validation
- Listing 6: MyTextWatcher attached to editText by addTextChangedListener
- Listing 7: Annotations for validation
- Listing 8: How to set up another layout
- Listing 9: Setup of instances with security
- Listing 10: usage of RepeatSafeToast
- Listing 11: Usage how to get time of launching main activity
- Listing 12: Usage of Aspect Faces in Java EE
- Listing 13: Usage of Aspect Faces in Android

Appendix D

References

- [1] Tomas Cerny, Michael J. Donahoo, and Eunjee Song. 2013. *Towards effective adaptive user interfaces design*. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS '13)*. ACM, New York, NY, USA, 373-380. DOI=10.1145/2513228.2513278
<http://doi.acm.org/10.1145/2513228.2513278>
- [2] Tomas Cerny, Karel Cemus, Michael J. Donahoo, and Eunjee Song. 2013. *Aspect-driven, Data-reflective and Context-aware User Interfaces Design*. In *Applied Computing Review, Vol. 13, Issue 4*, ACM, New York, NY, USA, 53-65. ISSN 559-6915
<http://www.sigapp.org/acr/Issues/V13.4/ACR-13-4-2013.pdf>
- [3] Tomas Cerny and Eunjee Song. 2011. *UML-based enhanced rich form generation*. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation (RACS '11)*. ACM, New York, NY, USA, 192-199. DOI=10.1145/2103380.2103420
<http://doi.acm.org/10.1145/2103380.2103420>
- [4] *Jak vypadá Android uvnitř*. *Android developpers [online]*. 31. Prosinec 2011 12:00 [cit. 2014-04-28].
<http://www.androidmarket.cz/android/jak-vypada-android-uvnitř-aneb-co-je-rom-kernel-bootloader-a-dalsi>
- [5] *Android Activity Lifecycle*. *Android Activity Lifecycle [online]*. 12/22/2011 [cit. 2014-04-28].
<http://www.mikestratton.net/2011/12/android-activity-lifecycle>
- [6] *Android vs IOS*. *Android vs IOS [online]*. November 11 2013 3:22 PM [cit. 2014-04-28].
<http://www.ibtimes.com/android-vs-ios-whats-most-popular-mobile-operating-system-your-country-1464892>

- [7] *October Android distribution numbers. Android police [online]*. November 11 2013 3:22 PM [cit. 2013/10/03].
<http://www.androidpolice.com/2013/10/03/october-android-distribution-numbers-more-people-get-jelly-bean-ice-cream-sandwich-and-gingerbread-usage-dips>
- [8] *Introduction To Android Mobile Operating System. Android Development, Tutorials [online]*. August 1, 2011 [cit. 2014-04-29].
<http://www.blogsaays.com/tutorial-part1-introduction-android-mobile-operating-system>
- [9] R. Kennard and J. Leaney. *Towards a general purpose architecture for ui generation. Journal of Systems and Software, 83(10):1896 - 1906, 2010*.
[http://metawidget.sourceforge.net/media/downloads/Towards a General Purpose Architecture for UI Generation.pdf](http://metawidget.sourceforge.net/media/downloads/Towards%20a%20General%20Purpose%20Architecture%20for%20UI%20Generation.pdf)
- [10] R. Kennard and S. Robert. *Application of software mining to automatic user interface generation. In SoMeT'08, pages 244 - 254, 2008*.
[http://metawidget.sourceforge.net/media/downloads/Application of Software Mining to Automatic User Interface Generation.pdf](http://metawidget.sourceforge.net/media/downloads/Application%20of%20Software%20Mining%20to%20Automatic%20User%20Interface%20Generation.pdf)
- [11] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. *Models@run.time to support dynamic adaptation. Computer, 42(10):44-51, Oct. 2009*.
- [12] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
- [13] I. R. Forman and N. Forman. *Java Reflection in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.

Appendix E

Content of attached CD

The content of CD is organized into the following files and folders:

- diploma_thesis – folder that contains diploma thesis in pdf format, but also there are source tex files, images and excel file with calculations, enterprise architekt files
- aspectFaces.zip – AOP framework
- conventionalApproach.zip – android application that was compared with AOP approach