Convalida: a Code Generation-based Field Validation Library for Android Applications

Wellington Costa Pereira, Paulo Henrique Maia[1]

[1]Group of Software Engineering and Distributed Systems (GESAD)
State University of Ceara (UECE)
1700 Dr. Silas Munguba Avenue, Fortaleza 60714-903, Brazil

wellington.pereira@aluno.uece.br, pauloh.maia@uece.br

Abstract. *A common, repetitive, and time-consuming task in the construction of Android applications is creating interface field validation. Although there are some tools that address that issue, they are not intuitive and require effort for configuration, which may hinder their use. This paper introduces Convalida, an annotation-based library that generates code automatically at compile time for field validation of Android applications, thus allowing the developer to focus on the implementation of business rules. A comparative study considering other field validation tool and a manual approach is also provided and shows that our library has improved the final code.*
Video: https://youtu.be/nRq8ryozZeE

Resumo. *Uma tarefa comum, repetitiva e demorada na construção de aplicativos Android é implementar a validação de campos de interface. Embora existam algumas ferramentas que abordem esse problema, elas não são intuitivas e exigem esforço para configuração, o que pode impedir seu uso. Este artigo apresenta a Convalida, uma biblioteca baseada em anotações que gera código automaticamente em tempo de compilação para validação de campos em aplicativos Android, permitindo que o desenvolvedor se concentre na implementação de regras de negócios. Um estudo comparativo considerando outras ferramentas de validação de campos e uma abordagem manual também é fornecido, demonstrando como a nossa biblioteca melhorou o código final.*

1. Introduction

The demand for Android applications is increasing due to the wide reach that the Android platform has been achieving over the last few years and, consequently, the growth of the user community. According to data from Net Marketshare[1] for April 2018, Android is the most used platform on the planet, with a percentage of 69,8%, reaching more than one billion mobile devices among smartphones and tablets.

Many Android applications, specially commercial ones, have forms that need to be populated by users to perform a type of registration. The inputted data should normally conform to some integrity rules, which can be applied to either the data layer, often using SQL constraints, or the presentation layer, acting directly in the form field values. The manual writing of field validation code can become a time-consuming and repetitive task,

[1]https://www.netmarketshare.com

since it is necessary to write the logic to validate the inputted data, as well as the display of the error messages to the user when some value does not satisfy the validation rule.

To mitigate that problem, this work proposes Convalida, a field validation library based on Java annotations and automatic code generation for Android applications. By using annotations, the library generates all pieces of code necessary for validating the mapped fields, allowing the developer to concentrate on only the implementation of the business rules and other aspects of the application. In the paper, we describe the library's structure and the available validation types and detail a comparative study including another field validation library and a manual approach.

2. Metaprogramming

Metaprogramming, or generative programming, is a programming technique in which a computer program is written to generate or manipulate another program or itself, in order to solve a given task [Cordy and Shukla 1992]. Among the different types of applications that can be developed through metaprograms, we can cite program transformation, detection and application of design patterns, program refactoring and code generators [De Oliveira et al. 2004] [Papotti et al. 2012]. In addition, it is constantly evolving and its principles are used at increasingly higher levels of abstraction, as in model-driven software engineering [Štuikys and Damaševičius 2008].

The metaprogramming allows reducing the number of lines of code in a software, thus promoting productivity. The technique also offers to the developer greater flexibility, managing new situations without the need for recompilation.

3. Related Work

In [Freitas and Maia 2015], the authors present JustBusiness, a Java framework that performs automatic Android code generation at compile time from annotations contained in the application business classes. JustBusiness creates automatically not only the required structure of an Android application, but also user interfaces and data persistence.

In [Parada et al. 2013], the authors propose a tool for generating Android code from UML models efficiently, based on Google's best practices for Android devices, such that the generated code is automatically optimized for the Android platform. The user can choose which optimization action will be applied, thus allowing to observe the impacts of the those actions in isolation.

Transfuse [Ericksen 2016] is an annotation-based framework that performs analysis and automatic code generation for the Android platform, aiming at a boilerplate reduction. Transfuse implements common metaprogramming techniques, such as code generation.

Android Saripaar[2] is an annotation-based library that was built based on the Apache Commons Validator[3] project. The library has a large set of validations for selection, text, numeric, and date fields. All validations are applied at runtime through reflection. The library also allows the programmer to create their own validations.

[2]https://github.com/ragunathjawahar/android-saripaar
[3]http://commons.apache.org/proper/commons-validator/

Data Binding Validation[4] is a declarative library through XML code, in which the programmer declares in the XML layout files the validations for several fields. The validations are applied through the Data Binding library[5], developed and maintained by Google, to make binding between XML layouts and Java code.

4. The Convalida Library

Convalida[6] is an annotation-based library that automatically generates code at compile time, developed using the Java programming language for the Android platform, aiming to streamline the field validation task in Android application projects. It is distributed under Apache 2.0 license.

Convalida exposes a set of annotations that can be used on Android input fields and, at compile time, these annotations, along with information passed by parameters, are processed and automatically generate validation codes according to the annotations applied to the input fields. In addition, the Convalida library can be used in conjunction with other libraries and frameworks those previously cited.

4.1. Structure

Convalida was designed with an independent and extensible module-based structure, enabling further uncoupling of code. The modules will be explained in more detail below.

Module Convalida Annotations contains all the annotations that can be used for developers to apply validations to the fields and get its result. Some annotations may contain parameters that serve as additional information to apply the validations.

Module Convalida Compiler is responsible for automatically generating all the codes to validate mapped fields with annotations, enabling the developer to focus on the implementation of business rules, thus increasing productivity in application development.

Module Convalida Validators contains all validation rules defintions. Each validation that is applied to the fields has its own validation class rule. This module also contains unit tests for each validation class rule.

Module Convalida Data Binding allows to apply the validations through XML layout definition instead of tradicitional Java annotations approach.

Module Convalida Kotlin Extensions is an alternative to use Convalida in a projects that is written in Kotlin programming language, by just using extension functions and no Java annotations nor code generation.

Module Convalida Runtime is the main module of library and centralizes the access to the other modules. This module also contains some predefined regular expressions that can be used for developers.

4.2. Using in a project

To use Convalida in an Android project, a developer must follow a set of steps that are detailed as follows.

[4]`https://github.com/Ilhasoft/data-binding-validator`

[5]`https://developer.android.com/topic/libraries/data-binding`

[6]`https://github.com/WellingtonCosta/convalida`

4.2.1. Add Convalida dependencies in the project

In order to facilitate the distribution of the Convalida library, its artifacts are available in Maven Central[7]. To use Convalida artifacts in the project, the developer needs to add the dependencies in the **build.gradle** file that is inside the application module folder, as shown in listing 4.2.1.

```
implementation "io.github.wellingtoncosta:convalida-runtime:3.1.0"
annotationProcessor "io.github.wellingtoncosta:convalida-compiler:3.1.0"
```

4.2.2. Map annotations to fields and methods

In order to automatically generate and apply the classes that contains the validation code to the respective fields, the library's users must map these fields to the validation annotations, their actions and its results callbacks as shown in the code shown in listing 4.2.2.

```
class SampleActivity extends Activity {
  // Mapping validations
  @Required(errorMessage = R.string.field_required) EditText nameField;

  @Email(errorMessage = R.string.invalid_email) EditText emailField;

  @Password(errorMessage = R.string.invalid_password) EditText passwordField;

  // Mapping actions and its results
  @ValidateOnClick Button validateButton;

  @ClearValidationsOnClick Button clearValidationsButton;

  @OnValidationSuccess void success() { ... }

  @OnValidationError void error() { ... }
}
```

After mapping the fields and actions, it is necessary to initialize the automatically generated class from the Convalida library to validate the fields mapped with the annotations as shown in listing 4.2.2. The generated class has the same name of source class with *FieldsValidation* suffix.

```
@Override protected void onCreate(Bundle bundle) {
  super.onCreate(bundle);
  setContentView(R.layout.activity_sample);
  SampleActivityFieldsValidation.init(this);
}
```

4.2.3. Build the project

This last step consists of cleaning the project and building it again. This step is necessary because automatic code generation happens during project compilation and not at runtime,

[7]https://search.maven.org/

like in other libraries. In this step all pieces of code are generated to validate the mapped fields with the validation annotations.

After performing all the steps listed above, if it is necessary to map new fields to be validated, this step must be performed again.

5. Comparative Study

In order to demonstrate the use of the Convalida library, we conducted a comparative study that contemplates the development of an application for registration of users and their respective contacts. We implemented the application using three different field validation approaches: the first one uses the Convalida library, the second one uses the Android Saripaar library, and in the last one the code for field validation was written manually by the programmer, with and without code reuse technique.

Figure 1 shows the class diagram of the application used in the comparative study. Although simple in terms of functionality, it allows to explore a significant amount of annotations available in the library proposed in this paper.

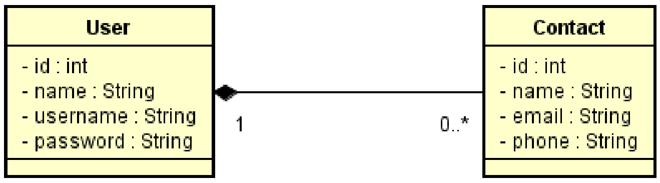

Figure 1. Application Class Diagram.

5.1. Evaluation

This section shows the results of the comparative study considering the number of lines of code in each implementation as comparison criteria. The source code for this comparative study is available on GitHub[8]. The criteria to count the lines of code was: annotations, Java extends and implements commands, methods and variables declaration, additional configurations, callbacks methods, condition and repetition instructions.

Table 1 shows the result of comparing the number of lines of code written by the programmer using the Android Saripaar library, Convalida library, written field validation codes manually with no code reuse and with code reuse, according to the application screen and the number of lines of code that were necessary to apply the validations to the fields application screens.

Table 1. Number of lines of code written by programmer

	Android Saripaar	Convalida	No code reuse	Code reuse
Login	20	5	20	5
User Register	23	7	23	7
Contact Register	21	6	21	6
Reused code	N/A	N/A	0	68

[8]`https://github.com/WellingtonCosta/comparative-study-convalida`

Finally, Table 2 shows the total number of lines of code between all approaches to implement the field validation of the application presented in this comparative study. In the sum of strategies for field validation with all codes written by the programmer, the total number of reused code was taken into account.

Table 2. Total lines of code between implementations

	Total of lines of code
Android Saripaar	69
Convalida	19
No code reuse	135
Code reuse	86

6. Conclusion

This paper presented Convalida, a library based on annotations and automatic code generation for field validation in Android applications. The main benefit is to boost the development process, allowing the developer to focus on implementing the application business rules rather than on manually writing field validation logic.

Despite its advantages, the library still has some limitations, such as no validation for selection fields, like radio buttons and checkboxes. As future work, we intend not only to implement other types of validation beyond those presented in this paper, but also to give the possibility for the developer to define their own specific validation logic in a simple and objective way, thus allowing the library to generate the validation code according to the rules defined by the developer.

References

Cordy, J. R. and Shukla, M. (1992). Practical metaprogramming. In *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research-Volume 1*, pages 215–224. IBM Press.

De Oliveira, A. A., Braga, T. H., de Almeida Maia, M., and da Silva Bigonha, R. (2004). Metaj: An extensible environment for metaprogramming in java. *J. UCS*, 10(7):872–891.

Ericksen, J. (2016). Transfuse: A compile-time metaprogramming solution for reducing boilerplate on google's android.

Freitas, F. and Maia, P. H. M. (2015). Just business: A framework for developing android applications using naked objects. In *2015 IX Brazilian Symposium on Components, Architectures and Reuse Software*, pages 11–20. IEEE.

Papotti, P. E., do Prado, A. F., and de Souza, W. L. (2012). An approach to support legacy systems reengineering to mdd using metaprogramming. In *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*, pages 1–10. IEEE.

Parada, A. G., Tonini, A. R., and de Brisolara, L. B. (2013). Geração automática de código android eficiente a partir de modelos uml. In *CIbSE*, pages 71–84.

Štuikys, V. and Damaševičius, R. (2008). Development of generative learning objects using feature diagrams and generative techniques. *Informatics in education*, 7:277–288.