

WEEK 16 – Design Patterns



By [plandoraproject <](#)

[https://plandora51897980.wordpress.com/author/plandoraproject/>](https://plandora51897980.wordpress.com/author/plandoraproject/)

[16. May 2021 <](#)



[https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/>](https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/)

[2 Comments <](#)

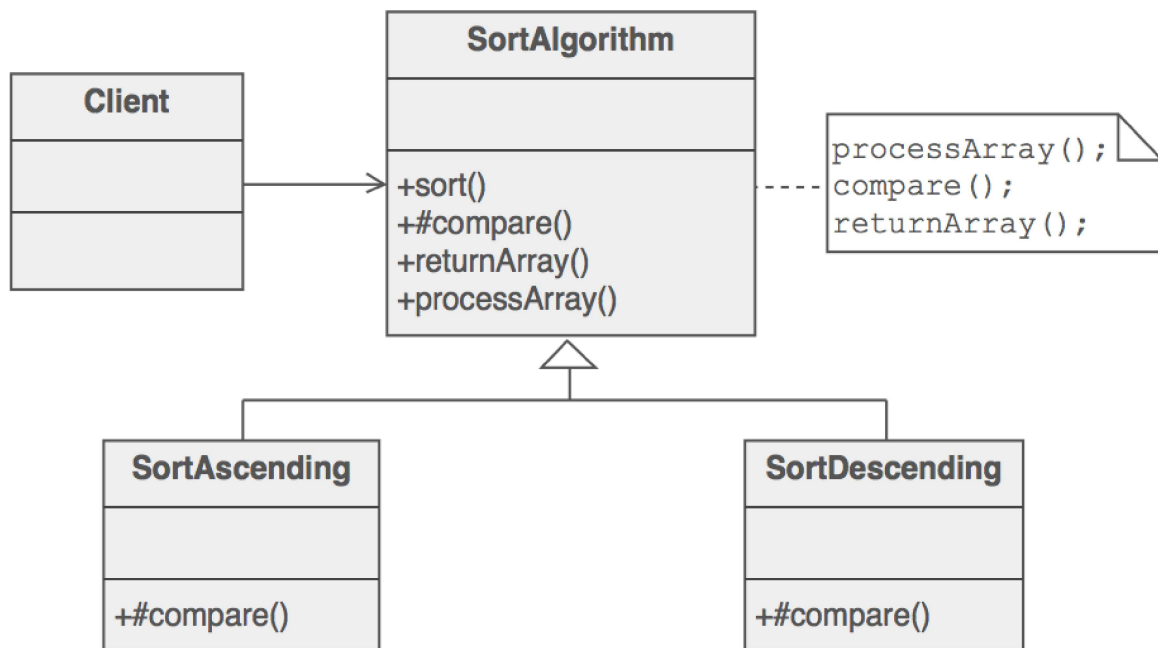


[https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/#comments>](https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/#comments)

This week we refactored our code using Design Patterns. Design Patterns describe the solutions for common problems independent of a specific programming language. In this case we used the so-called Template Method Pattern to find a common and expandable implementation for all activities in which a client-side validation of user input is required.

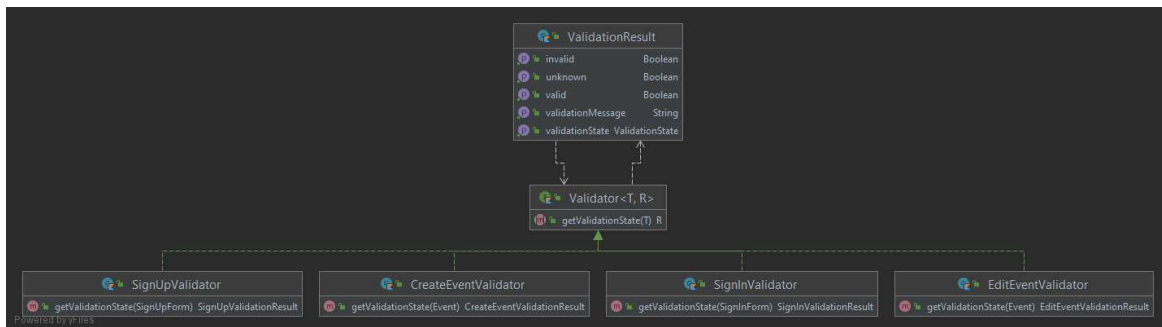
The logic for validation has been implemented inconsistently so far and was strongly coupled to the corresponding UI components. This circumstance made it almost impossible to reuse and test the code.

The Template Method Pattern is part of the so-called Behavioral Design Patterns, which generally describe the way how different classes communicate with each other. The Template Method Pattern provides a base class which declares one or more placeholders in the form of method signatures. These placeholders are implemented by any number of sub-classes. An exemplary UML Diagram for a sorting algorithm implemented with the Template Method Pattern is shown below:

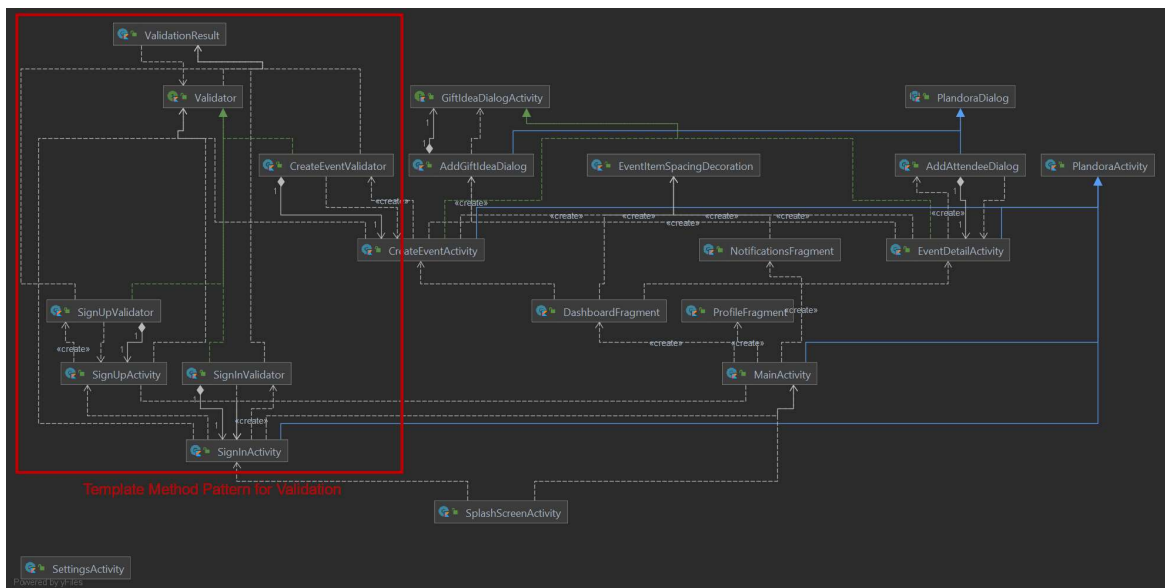
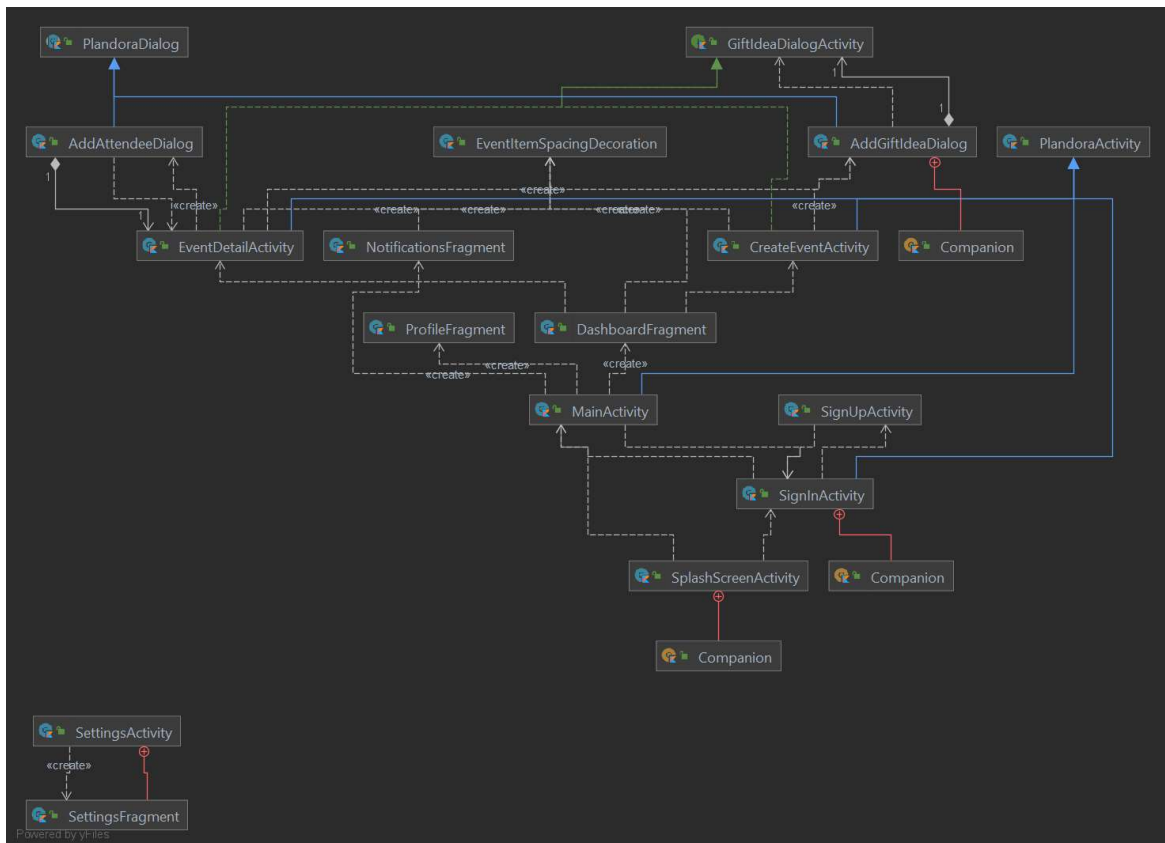


Source: [<https://sourcemaking.com/design_patterns<https://sourcemaking.com/design_patterns>](https://sourcemaking.com/design_patterns) (05/16/2021)

In our case the **SortAlgorithm** Class corresponds to a generic **Validator** Class, while the different sub-classes (**SortAscending**, **SortDescending**) correspond to the implementation of validators. For instance, we need validation for the sign up form and for the creation of a new event. The following image shows how we implemented the pattern on the example of the described issue:



In addition to this, the next two images show the changes of the whole* UML diagram caused by this measure.



* only the relevant part ist displayed, see [SAD < https://github.com/Honrix/PlandoraDocumentation/blob/main/SAD/SAD.md>](https://github.com/Honrix/PlandoraDocumentation/blob/main/SAD/SAD.md) for more information

On the example of the sign up form, the following links show that, thanks to the refactoring, the validation is more decoupled from the UI Component (SignUpActivity).

- Before <
<https://github.com/nf3lix/Plandora/blob/9e51bc4f5dc98432ab5af6debedc04791f7d9312/app/src/main/java/com/plandora/activity/launch/SignUpActivity.kt#L35>>
- After <
<https://github.com/nf3lix/Plandora/blob/c5dba1f941257877b5a8957e5e6a171921018284/app/src/main/java/com/plandora/activity/launch/SignUpActivity.kt#L32>>

The following code snippet shows the Validator base-class. Since each validator works with a different data model and returns different results, the sub-classes must specify the parameters T & R.

```
interface Validator<T: Any, R: ValidationResult> {  
  
    enum class ValidationState {  
        VALID, INVALID, UNKNOWN  
    }  
  
    fun getValidationState(data: T): R  
}
```

The validator sub-class for the sign up form is shown below. It implements the Validator-Interface and defines an abstract sub-class for possible results returned by the getValidationState-Function.

```

class SignUpValidator : Validator<SignUpForm,
SignUpValidator.SignUpValidationResult> {

    override fun getValidationState(data: SignUpForm):
SignUpValidationResult {
        return when {
            // returns result depending on the sign up data
        }
    }

    sealed class ValidationResults {
        // SignUpValidationResult sub-classes
    }

    abstract class SignUpValidationResult(val state:
Validator.ValidationState, val message: String) :
ValidationResult(state, message)

}

```

The benefits of these changes are the improved extendibility and encapsulation, which also leads to an better testability.


Advertisements


Occasionally, some of your visitors may see an advertisement here, as well as a [Privacy & Cookies banner < https://en.support.wordpress.com/cookie-widget/>](#) at the bottom of the page. You can hide ads completely by upgrading to one of our paid plans.


UPGRADE NOW < [HTTPS://WORDPRESS.COM/PLANS/183464470/?FEATURE=NO-ADVERTS&UTM_CAMPAIGN=REMOVEADSNOTIVE](https://wordpress.com/plans/183464470/?FEATURE=NO-ADVERTS&UTM_CAMPAIGN=REMOVEADSNOTIVE)>

DISMISS MESSAGE

Teilen mit:

 Press This < <https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/?share=press-this&nb=1>>

 Twitter < <https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/?share=twitter&nb=1>>

 Facebook < <https://plandora51897980.wordpress.com/2021/05/16/week-16-design-patterns/?share=facebook&nb=1>>

[Customise buttons < https://jetpack.com/redirect/?source=calypso-marketing-sharing-buttons&site=plandora51897980.wordpress.com>](https://jetpack.com/redirect/?source=calypso-marketing-sharing-buttons&site=plandora51897980.wordpress.com)

Reblog

Like

Be the first to like this.

Related

WEEK 17 - Metrics

31. May 2021

In "Allgemein"

WEEK 15 -

Refactoring

9. May 2021

In "Allgemein"

WEEK 14 - Testing

1. May 2021

In "Allgemein"

 [Edit < https://wordpress.com/post/plandora51897980.wordpress.com/270>](https://wordpress.com/post/plandora51897980.wordpress.com/270)

← [WEEK 15 - Refactoring](#) ≤ <https://plandora51897980.wordpress.com/2021/05/09/week-15-refactoring/>

→ [WEEK 17 - Metrics](#) ≤ <https://plandora51897980.wordpress.com/2021/05/31/week-17-metrics/>

