Creative DSLs in Kotlin

Daniel Gronau

Table of Contents

Creative DSLs in Kotlin

Published by me.

Production Editor: Daniel Gronau
Copy Editor: Daniel Gronau
Technical Editor: Daniel Gronau
Cover and Interior Design: Daniel Gronau

Library of Congress Cataloguing-in-Publication Data:
ISBN: XXX-X-XXX-XXXXX-X

Dedication

To my grandma Waltraud Reuschel, who taught me to read.

Acknowledgements

Writing a book is easy. Becoming a person able to write a book is the hard thing. That’s why I want to thank my parents Eva and Wolfgang Gronau, together with all the excellent teachers I had.

I also want to thank my friends Ernst Salzmann and Falk Wallis, and the members of the Byte-Welt forum, whose feedback was invaluable for this book.

And last but not least, I want to thank all the heros who provide free knowledge and awesome open source tools like the Kotlin language itself, making the world a better place.

Preface

Why another book about DSLs in Kotlin?

Kotlin is very suitable to write DSLs and accordingly there exists already a lot of literature about the topic. However, in my opinion many existing books fall short in two ways:

- They focus too much on "toy examples" that give little insight in the process of writing DSLs in practice
- They present solutions in a "vacuum", without highlighting how an effective design process looks like, which requirements, constraints and limitations apply, and which trade-offs were made

This book tries to address these gaps. It contains a check-list of requirements, which includes less obvious, but nevertheless important aspects. It further shows, how designing DSLs can be structured. Further, the presented DSLs progress from "boring" default examples to more and more challenging and unwieldy cases, and culminate in a discussion of some complex and successful real world DSLs.

Another goal of this book is to avoid presenting DSLs as isolated case studies, but to attempt a classification of DLS. Like design patterns established a common language for field-tested coding solutions, there should be a widely accepted terminology for classifying DSLs as well, and this book wants to contribute to this effort.

Who shouldn’t read this book (yet)?

Writing DSLs can be challenging, usually it isn’t a good introduction to a host language. That’s why this book assumes that you are already somewhat familiar with Kotlin, and - to a lesser degree - with Java.

There *is* actually a chapter describing many language features of Kotlin, but it is focussing on their possible use in DSLs - so again, this won’t be enough to learn the language.

So, if you don’t feel comfortable in Kotlin yet, take your time and explore the language first, it will save you from a lot of confusion. This book won’t go anywhere in the meantime, it can wait until you can get the most out of it.

Prerequisites

To try out code samples from this book or the associated project https://github.com/creativeDsls, you can use any IDE capable of running Kotlin (obviously IntelliJ IDEA will have an edge regarding Kotlin support), or you can go to https://play.kotlinlang.org/ for an online sandbox. The Kotlin language version used is 1.7. If not stated otherwise, when discussing Java, language level 11 is assumed.

Typographical Conventions

This book uses a few typographical conventions to structure its content.

Inlined code snippets are shown like this: `val answer = 42`

Tips, warnings etc. are shown as follows:

When the white frost comes, do not eat the yellow snow.

First of all, keep him out of the light, he hates bright light, especially sunlight, it'll kill him. Second, don't give him any water, not even to drink. But the most important rule, the rule you can never forget, no matter how much he cries, no matter how much he begs, never feed him after midnight.

Example code is shown like this, and may include the file name or lines with numbered remarks:

src/main/kotlin/personDemo/Person.kt

```
import java.time.ZonedDateTime
import java.time.temporal.ChronoUnit.YEARS

data class Person(val firstName: String, val lastName: String, val age: Int) {①
   constructor(firstName: String, lastName: String, dateOfBirth: ZonedDateTime):②
       this(firstName, lastName, YEARS.between(dateOfBirth, ZonedDateTime.now()).
toInt())
}
```

① This is the **main constructor**. For data classes, it also describes the structure of the object and its `toString()` representation.

② **Secondary constructors** allow to initialize the object in other ways, however they must delegate to the main constructor.

Additional information may be presented as a sidebar:

The name "Kotlin"

"Kotlin" is a small Russian island in the Baltic Sea. Naming languages or projects after islands has been a tradition in the Java ecosystem. Beside Java itself, there exists e.g. project Lombok, and the Ceylon language.

Quotes look like this:

> Code is like humor. When you have to explain it, it's bad.
>
> — Cory House

Feedback

I’m always thankful for feedback from my readers. Constructive criticism helps to improve this book. As every book, it contains mistakes, omissions, unclear formulations and - as I’m painfully aware as a non-native speaker - bad grammar.

To send me feedback, e-mail me at feedback@creativedsls.com.

Chapter 1. Introduction

1.1. What is a DSL?

> A Domain-Specific Language (DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem.
>
> — Martin Fowler, Domain-Specific Languages Guide

The intention of a DSL is to make a certain domain more accessible, to make it easier to read and write, to avoid mistakes, and sometimes to follow established standards (e.g. SQL for database access).

Typical examples include DSLs modelling business logic, e.g. trades for a financial company, and DSLs at the boundaries of the system, which simplify things like database access, serialization, web connectivity, UI. Another common application for DSLs is testing.

1.2. Internal and External DSLs

This books discusses DSLs which are embedded into Kotlin, and are therefore limited to the existing expressions of the language. These are *internal DSLs* (or "embedded DSLs). One major advantage is that these DSL don't need special treatment, there are no extra steps like reading and parsing files, so they fit seamless with the rest of the code. An edge case are DSLs where the DSL is realized entirely inside Strings, like regular expressions: While they are technically internal DSLs, they feel and behave more like external DSLs, because the "embedding" in the language is very shallow, and therefore they don't benefit from compile-time type checks or code completion like other internal DSLs do.

One disadvantage of internal DSLs is the limitation of the syntax inherited by the host language. Kotlin allows great freedom for DSL design, especially compared to Java. Nevertheless it is still possible that the language is not expressive enough to design the DSL you need. In this case *external DSLs* are an option: They have their own rules and syntax, and requiring a parser etc. leads to a larger overhead. However, writing external DSLs has become much easier in the last years by new libraries and frameworks, and improved tooling.

1.3. Code Generation for Internal DSLs

A common problem in DSL design is combinatical explosion: Take for example a DSL for physical units, and all the possible results when multiplying or dividing them. In order to achieve a pleasing syntax, but avoid illegal conversions you might have to write lots of boilerplate code. In these cases, code generation can help to get the job done, even if it seemed hopeless at the first glance.

Chapter 2. Source Codes

1500'lerden beri kullanılmakta olan standard Lorem Ipsum metinleri ilgilenenler için yeniden üretilmiştir. Çiçero tarafından yazılan 1.10.32 ve 1.10.33 bölümleri de 1914 H.Rackham çevirisinden alınan İngilizce sürümleri eşliğinde özgün biçiminden yeniden üretilmiştir.

Editable.java

```
public interface Editable{

    void useAsciidocFX();

}
```

app.rb

```
require 'sinatra'

get '/hi' do ①
  "Hello World!" ②
end
```

① Hooks `/hi` path when get request

② Returns "Hello World!"

Chapter 3. Asciidoc Table

Lorem Ipsum pasajlarının birçok çeşitlemesi vardır. Ancak bunların büyük bir çoğunluğu mizah katılarak veya rastgele sözcükler eklenerek değiştirilmişlerdir. Eğer bir Lorem Ipsumpasajı kullanacaksanız, metin aralarına utandırıcı sözcükler gizlenmediğinden emin olmanız gerekir. İnternet'teki tüm Lorem Ipsum üreteçleri önceden belirlenmiş metin bloklarını yineler.

Table 1. Table Title (Optional)

abcdefq	**abcdefq**	**abcdefq**	**abcdefq**
abcdefq	abcdefq	abcdefq	abcdefq
abcdefq	abcdefq	abcdefq	abcdefq

Chapter 4. Using UML Diagrams

You can usePlantUML extension

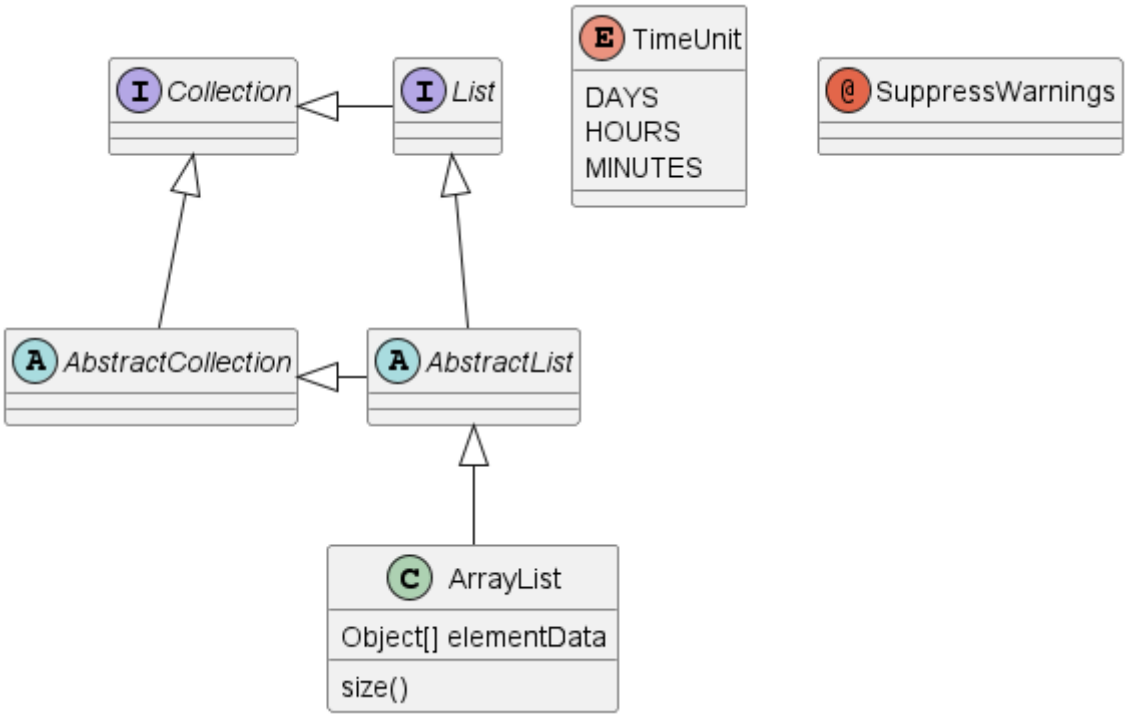


Figure 1. UML http://plantuml.sourceforge.net/

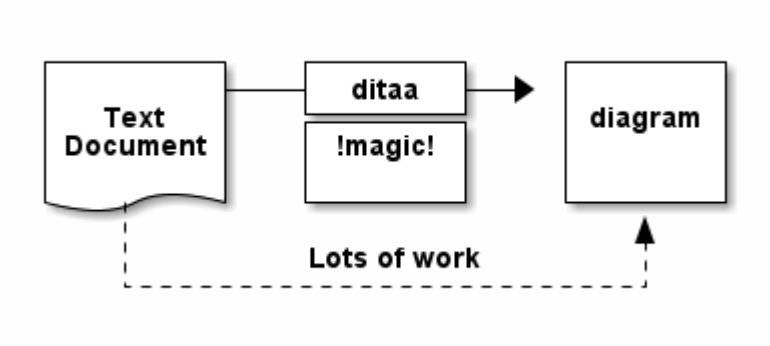


Figure 2. UML http://plantuml.sourceforge.net/ditaa.html

GraphViz has crashed

Figure 3. UML http://plantuml.sourceforge.net/ditaa.html

Chapter 5. Using Mathematical Formulas

MathJax is an open source JavaScript display engine formathematics that works in all browsers. In addition to png output, you can produce svg output also. Just change the extension.

$$\dot{x} = \sigma(y - x)$$

$$\dot{u} = \rho x - y - xz$$

$$\dot{z} = -\beta z + xyz$$

Tex Example

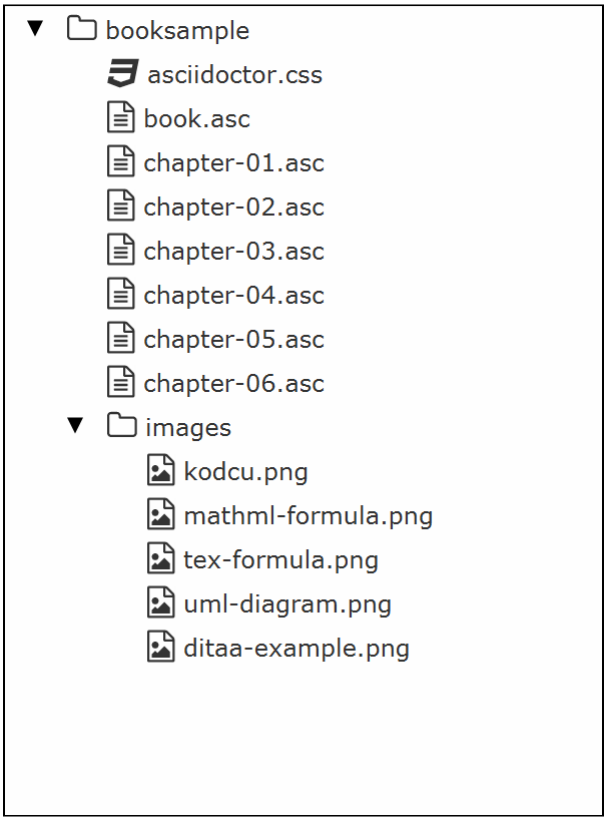
You can useTex orMathML languages for describing mathematical formulas in AsciidocFX. AsciidocFX converts this textual formulas as png image.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

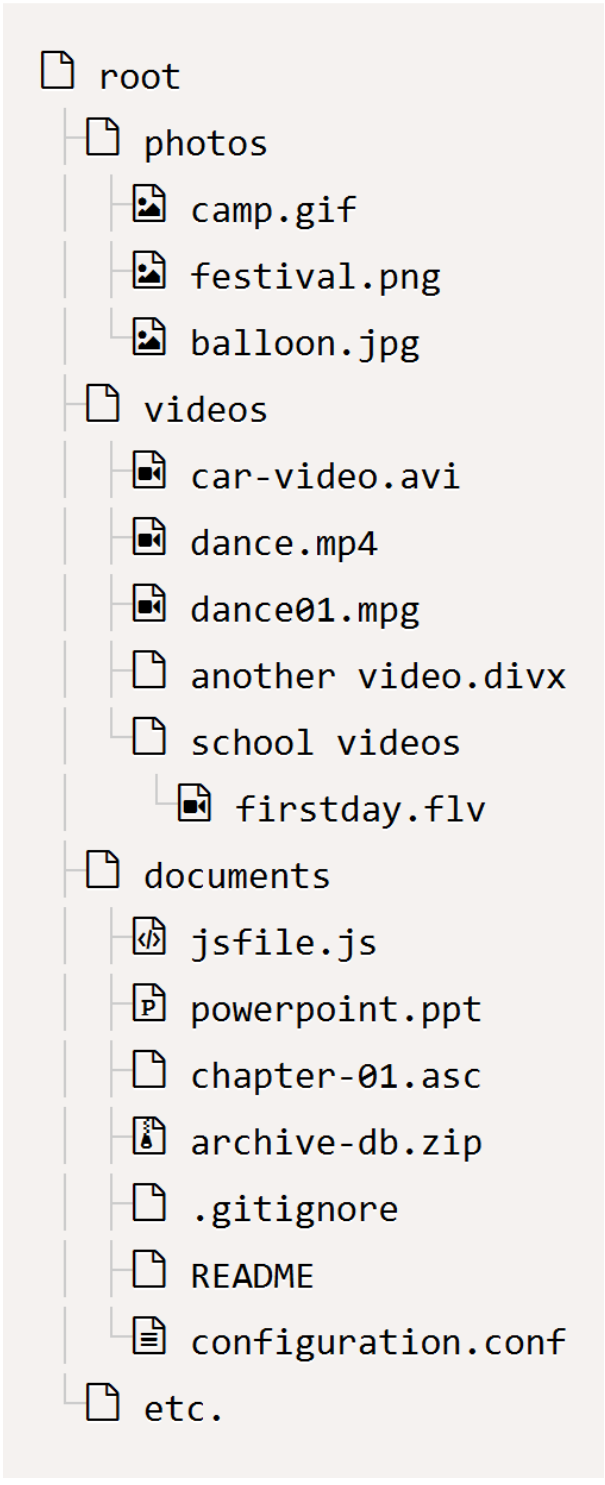
MathML Example

Chapter 6. Filesystem Tree Viewers

You can use filesystem viewer extension to demonstratefilesystem tree. We have two type of fs tree style.



Filesystem Tree

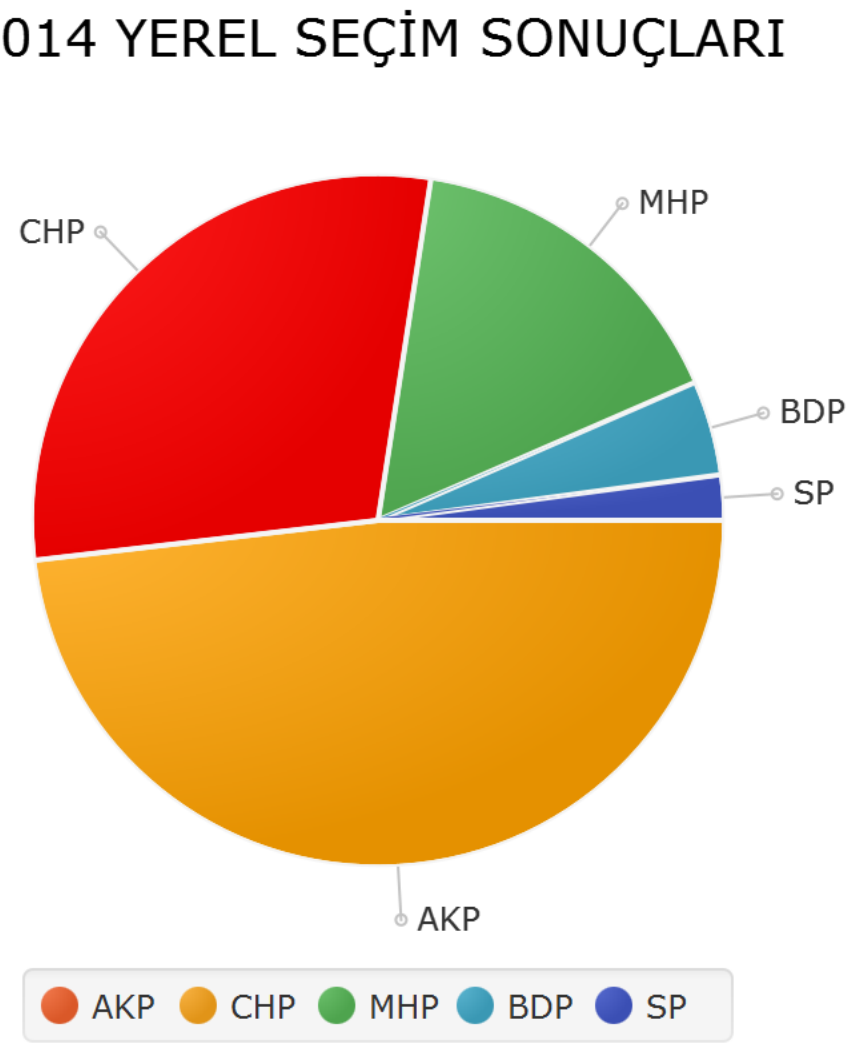


Filesystem Tree

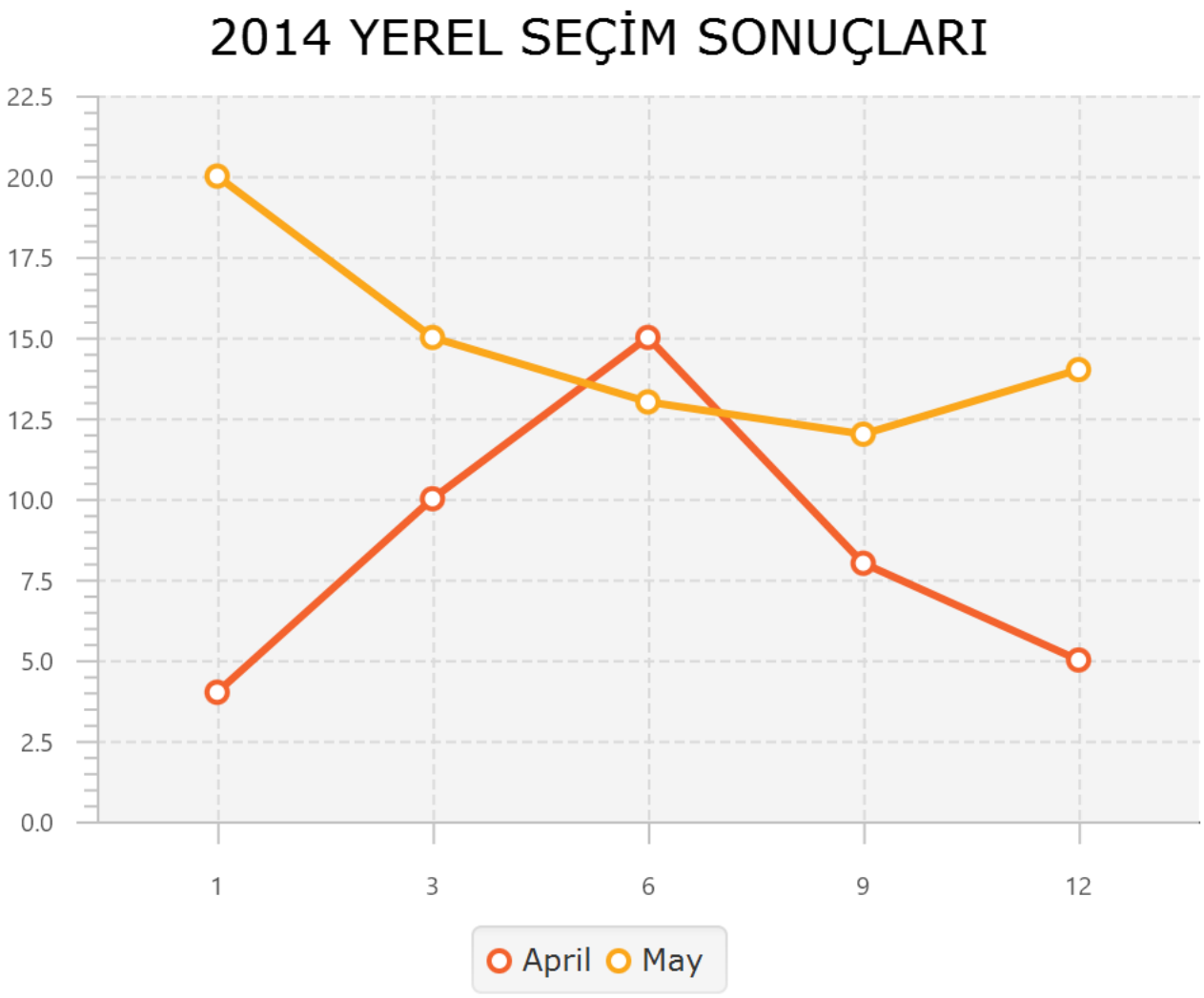
Chapter 7. AsciidocFX Charts

JavaFX has 8 kind of Chart component and AsciidocFX supports all of them. To see all available options please look at chart options

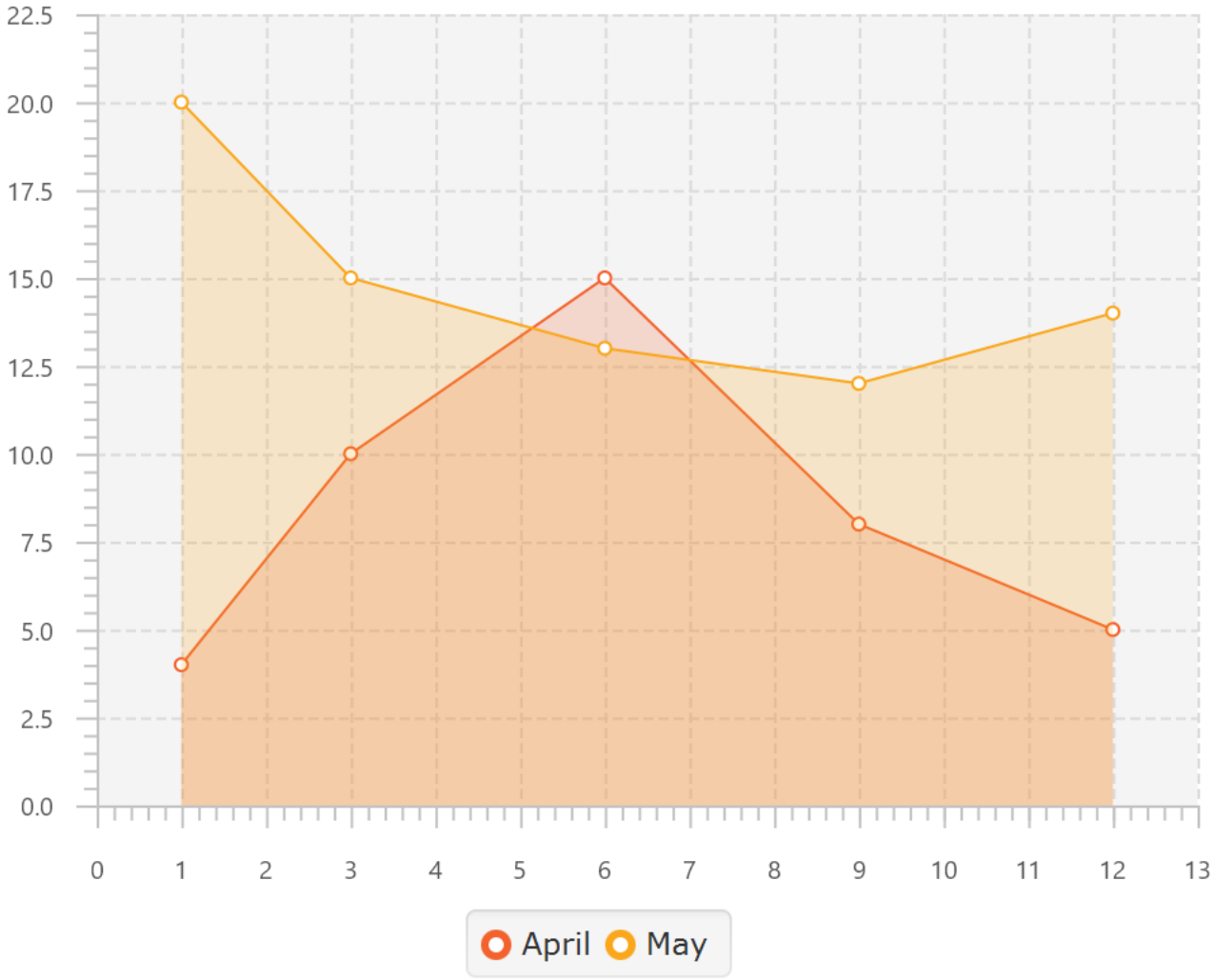
7.1. Pie Chart



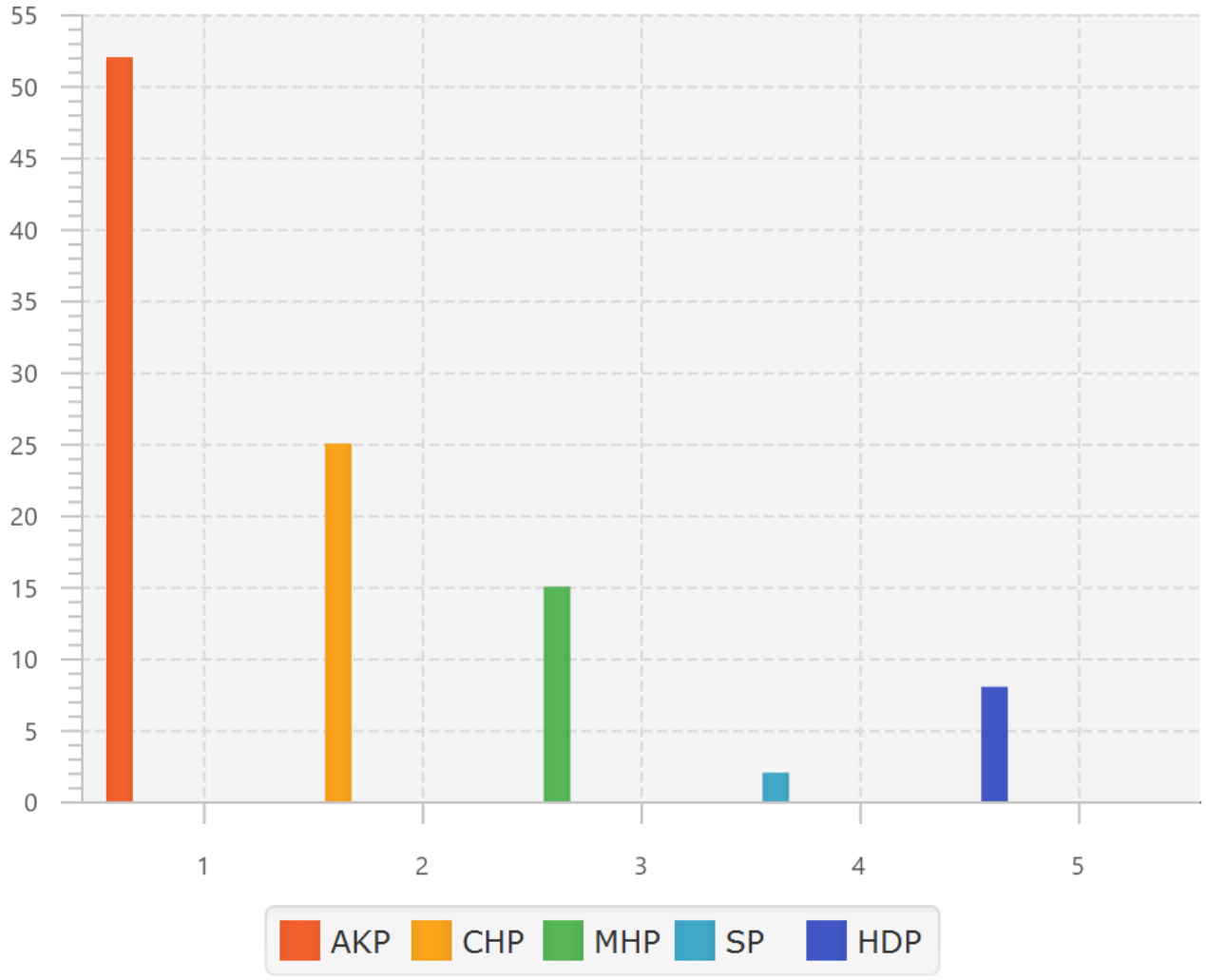
7.2. Line Chart



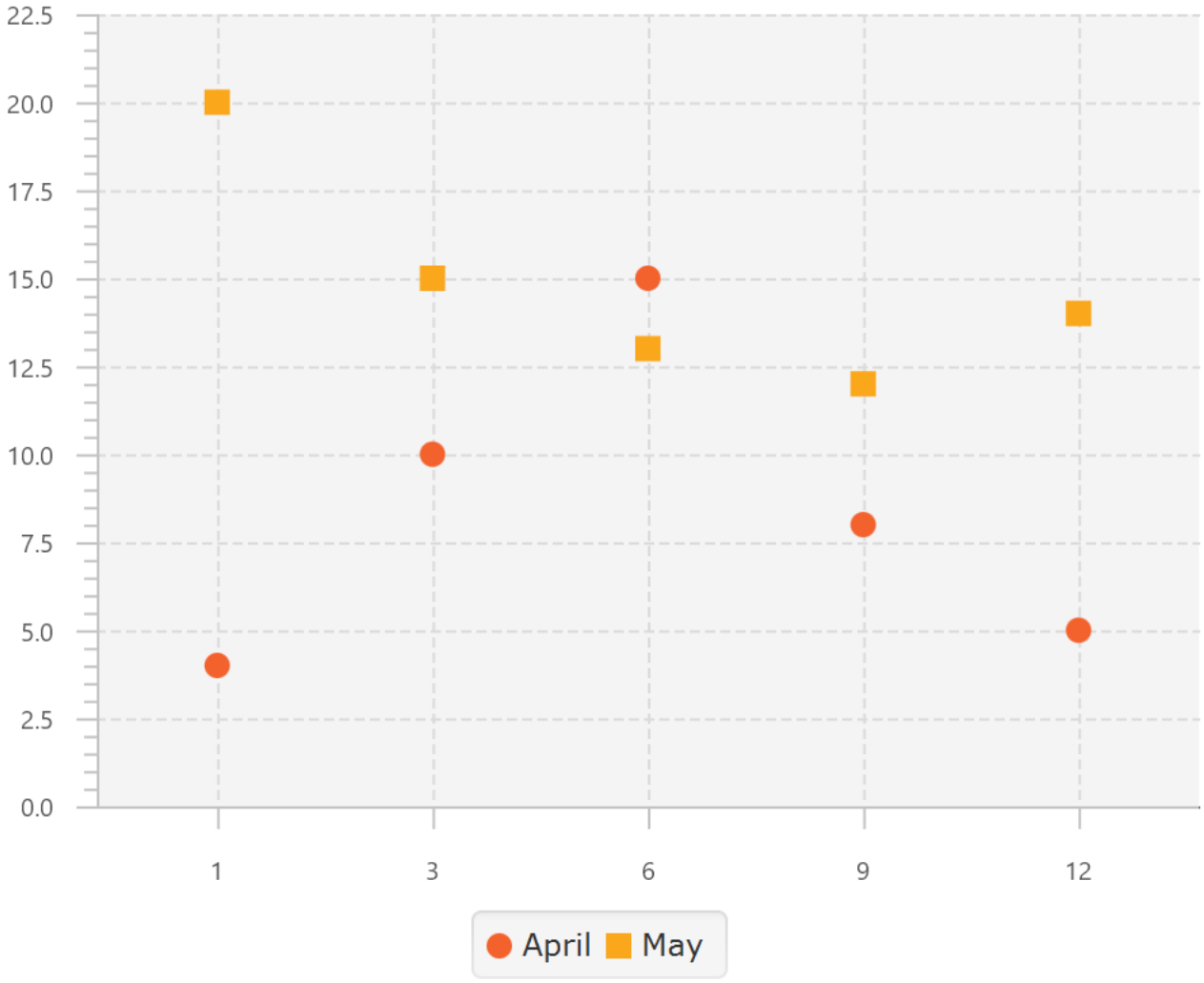
7.3. Area Chart


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
April
May

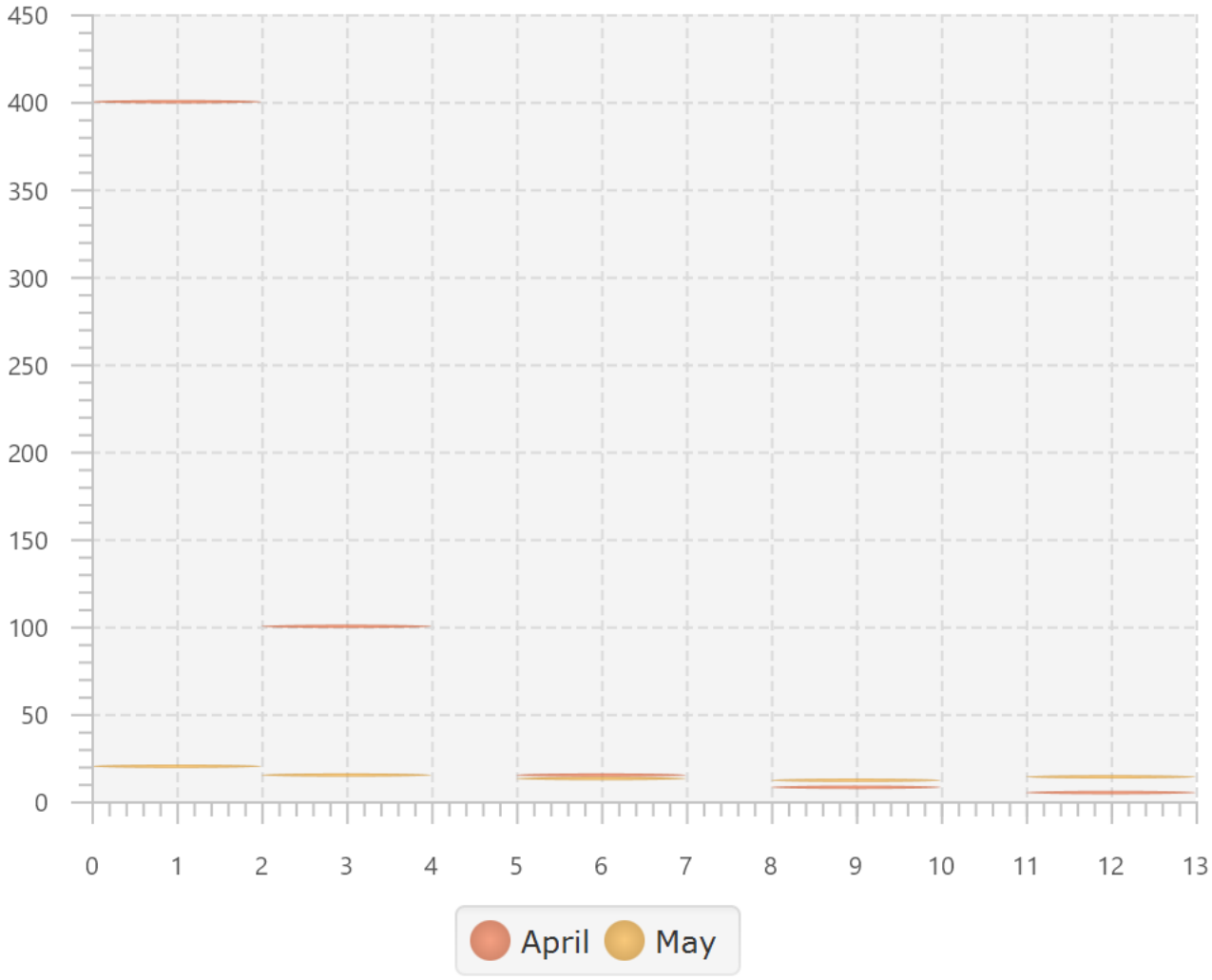
7.4. Bar Chart



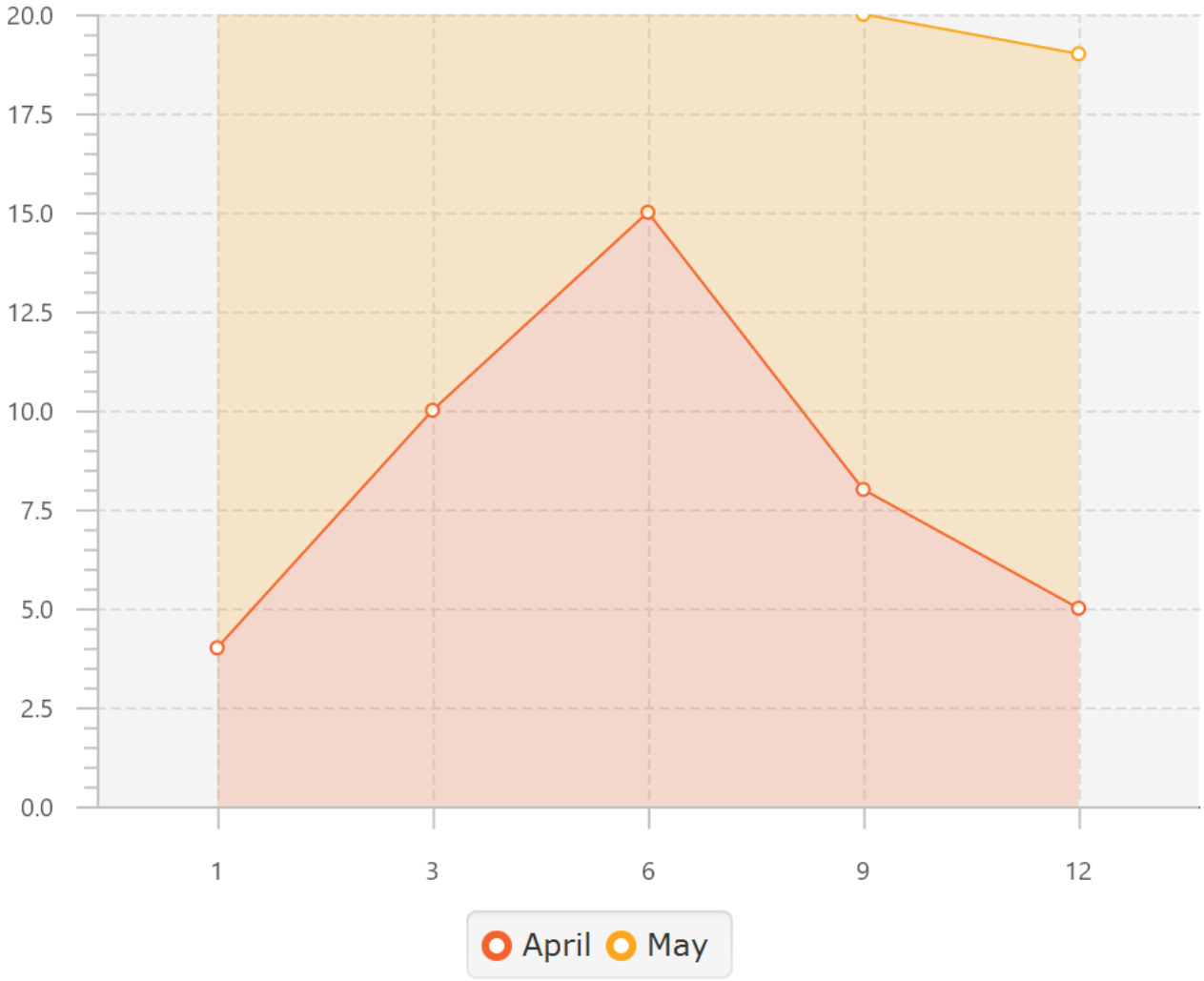
7.5. Scatter Chart



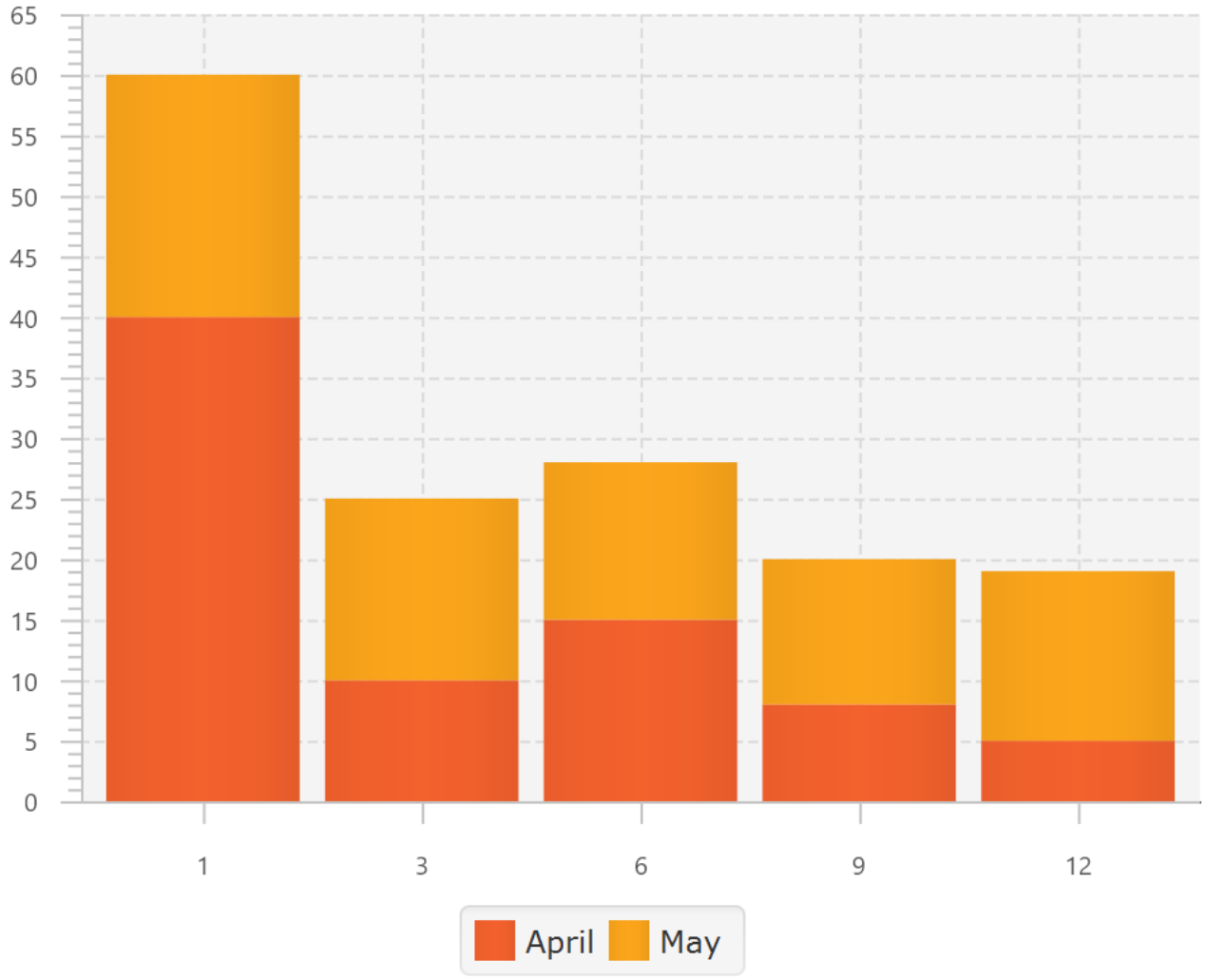
7.6. Bubble Chart



7.7. Stacked Area Chart



7.8. Stacked Bar Chart


65
60
55
50
45
40
35
30
25
20
15
10
5
0
1
3
6
9
12
April
May

Chapter 8. Using ditaa Diagrams

You can useditaa syntax to draw diagrams:

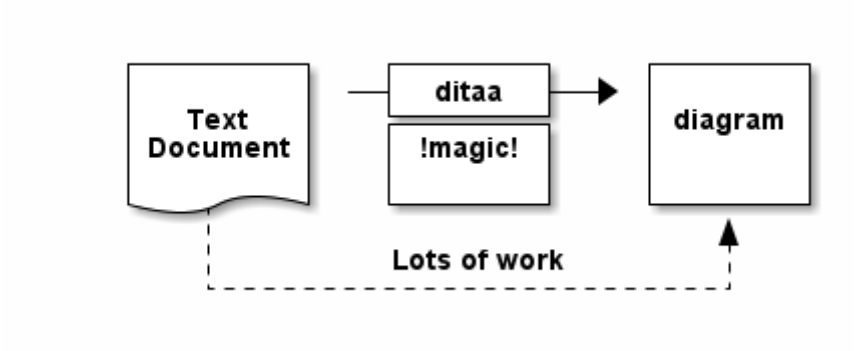


Figure 4. Ditaa http://ditaa.sourceforge.net/

Example Bibliography

The bibliography list is a style of AsciiDoc bulleted list.

Books

- [taoup] Eric Steven Raymond. 'The Art of Unix Programming'. Addison-Wesley. ISBN 0-13-142901-9.
- [walsh-muellner] Norman Walsh & Leonard Muellner. 'DocBook - The Definitive Guide'. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.

Index