Vladimir Stankovic

Sit down, relax, mix yourself a drink and enjoy the show ...

About

June 9, 2016 · DESIGN-PATTERNS JAVA STEP-BUILDER-PATTERN OOP

# Design patterns: Step builder

## Table of contents:



### Introduction

I have recently decided to use Amazon SES API in order to be able to send emails to my **Microservices Weekly** subscribers. What prompted me to use Amazon SES API is its price. It's really cheap. However, the client Java API provided by Amazon is not so simple to interact with, so I decided to create a **small wrapper** around their API.

To make long story short, the main purpose here is to share my experience with a less well-known derivate of Builder pattern - Step Builder pattern.

The Step Builder pattern is an object creation software design pattern. It's not so commonly mentioned in popular readings about design patterns. Honestly, I've heard about this pattern only recently from my collegue **@mikeladev**.

The Step Builder pattern offers some neat benefits when you compare it to traditional builder pattern. One of the main Step Builder pattern benefits is providing the client with the guidelines on how your API should be used. It can be seen as a mixture of a builder pattern and a state machine and in fact, this pattern is often referred to as a wizard for building objects.

### Pros and cons

**Pros**

1. User guidance for your API through object creation process step by step.
2. API User can call builder's build() method once the object is in the consistent state.
3. Reduced opportunity for creation of inconsistent object instances.
4. Sequencing initialization of mandatory fields.
5. Fluent API.
6. No need for providing validate() method for field validation.

**Cons**

1. Low readability of code needed to implement the pattern itself.
2. No eclipse plugin to help with code generation. (On the other hand, there are plenty of code generators for Builder pattern generator).

### Code-walkthrough

Since the Step Builder pattern is a creational design pattern, let's focus on its purpose - creation of objects.

Example of API usage is shown below:

```
Email email =
Email.builder().from(EmailAddress.of("Microservices Weekly <mw@microservicesweekly.com>"))
    .to(EmailAddress.of("svlada@gmail.com"))
    .subject(Subject.of("Subject"))
    .content(Content.of("Test email"))
    .build();
```

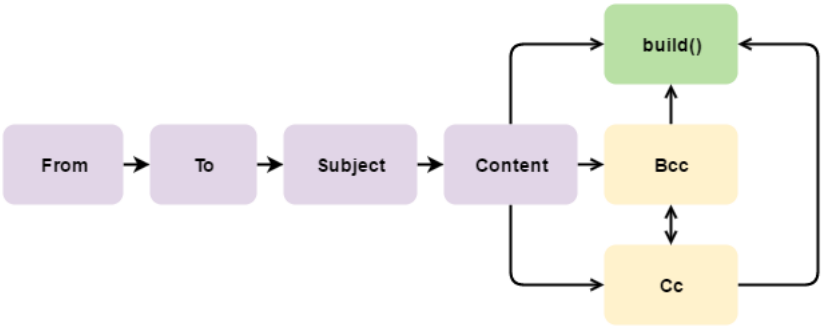
Now, let's see how API is enforcing initialiation of object in the pre-defined order.

Vladimir Stankovic

Sit down, relax, mix yourself a drink and enjoy the show ...

About

The following image is the graphical representation of the state machine for constructing Email object with the Step Builder pattern (mandatory values are marked purple and optional values are yellow):




Rules of thumb for implementation:

1. Add dependencies to your class. It's recommended to add private modifier to class attributes.
2. Define creational steps as inner interfaces in your base class.
3. Each creational step should return next step (interface) in chain.
4. Final step should be interface called "Build" which will provide build() method.
5. Define one inner static Builder class which implements all of defined steps.
6. Implement step interface methods.

## Source code

Complete Example of Step by Step builder pattern:

```
public class Email {
    private EmailAddress from;
    private List<EmailAddress> to;
    private List<EmailAddress> cc;
    private List<EmailAddress> bcc;
    private Subject subject;
    private Content content;

    public static FromStep builder() {
        return new Builder();
    }

    public interface FromStep {
        ToStep from(EmailAddress from);
    }

    public interface ToStep {
        SubjectStep to(EmailAddress... from);
    }

    public interface SubjectStep {
        ContentStep subject(Subject subject);
    }

    public interface ContentStep {
        Build content(Content content);
    }

    public interface Build {
        Email build();
        Build cc(EmailAddress... cc);
        Build bcc(EmailAddress... bcc);
    }

    public static class Builder implements FromStep, ToStep, SubjectStep, ContentStep, Build {
        private EmailAddress from;
        private List<EmailAddress> to;
        private List<EmailAddress> cc;
        private List<EmailAddress> bcc;
        private Subject subject;
        private Content content;

        @Override
        public Email build() {
            return new Email(this);
        }
```

Vladimir Stankovic

Sit down, relax, mix yourself a drink and enjoy the show ...

About

```
        @Override
        public Build cc(EmailAddress... cc) {
            Objects.requireNonNull(cc);
            this.cc = new ArrayList<EmailAddress>(Arrays.asList(cc));
            return this;
        }
        @Override
        public Build bcc(EmailAddress... bcc) {
            Objects.requireNonNull(bcc);
            this.bcc = new ArrayList<EmailAddress>(Arrays.asList(bcc));
            return this;
        }
        @Override
        public Build content(Content content) {
            Objects.requireNonNull(content);
            this.content = content;
            return this;
        }
        @Override
        public ContentStep subject(Subject subject) {
            Objects.requireNonNull(subject);
            this.subject = subject;
            return this;
        }
        @Override
        public SubjectStep to(EmailAddress... to) {
            Objects.requireNonNull(to);
            this.to = new ArrayList<EmailAddress>(Arrays.asList(to));
            return this;
        }
        @Override
        public ToStep from(EmailAddress from) {
            Objects.requireNonNull(from);
            this.from = from;
            return this;
        }
    }

    private Email(Builder builder) {
        this.from = builder.from;
        this.to = builder.to;
        this.cc = builder.cc;
        this.bcc = builder.bcc;
        this.subject = builder.subject;
        this.content = builder.content;
    }

    public EmailAddress getFrom() {
        return from;
    }

    public List<EmailAddress> getTo() {
        return to;
    }

    public List<EmailAddress> getCc() {
        return cc;
    }

    public List<EmailAddress> getBcc() {
        return bcc;
    }

    public Subject getSubject() {
        return subject;
    }

    public Content getContent() {
        return content;
    }
}
```

## Eclipse plug-in

So far, I haven't found a plug-in for Eclipse that provides Step Builder code generation feature.

I have created a github repository with the intention of creating an Eclipse plug-in that will provide support for Step Builder pattern generation: **https://github.com/svlada/alyx**

Vladimir Stankovic

Sit down, relax, mix yourself a drink and enjoy the show ...

About

ALSO ON SVLADA

**Proxy Ajax requests, Curl and Symfony 2**

3 comments • 3 years ago•

Frank Möller — Another approach is (if you have admin permissions on your server) using mod_proxy. Then you don't need an own proxy

**Require.js Optimization – part2**

1 comment • 3 years ago•

Alex Mills — thanks, this was very valuable. the one thing I can say to help others out - in order for r.js to include ALL the files you want to

**Public key authentication with Java over SSH**

8 comments • 2 years ago•

Vladimir Stanković — Hey John,Glad you liked the article Actually StrictHostKeyChecking at

**Override jQuery.ajax handler**

1 comment • 3 years ago•

Jonathan Wright — deferred.pipe is deprecated in favor of deferred.then.$(selector).live is deprecated in favor of $(selector) on Either way