# Understanding DSLs

Andrejs Doronins

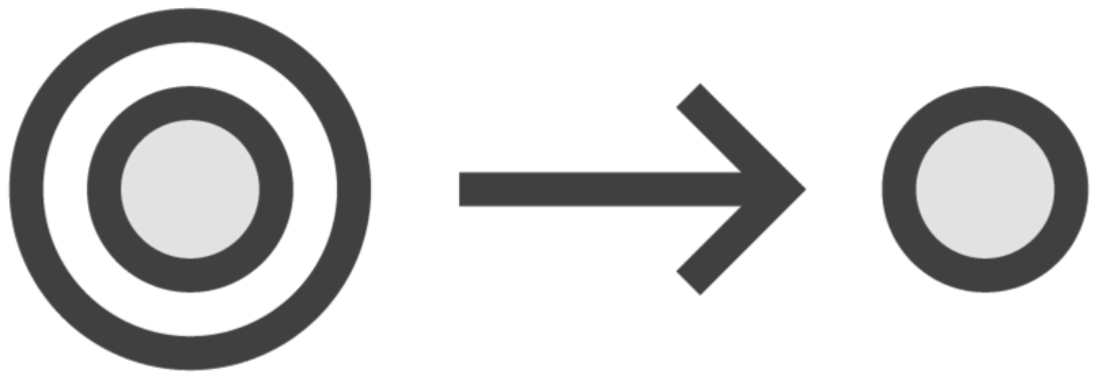
## DSL with a Fluent Interface

Method chaining

```
RestAssured.get(“api/url…”)
          .then()
          .assertThat()
              .statusCode(200)
          .and()
              .contentType(ContentType.JSON);
```

## DSL – Domain Specific Language

**A language focused on a particular domain and its keywords reflect that domain**



General programming languages:

- e.g. Java, C# or Python
- use to build (almost) anything
- wide scope

DSLs:

- specialized
- e.g. SQL – focus on DBs

```
SELECT * FROM World ORDER BY Country DESC;
```

✓ Specialized

✗ Not (very) fluent

```java
public T then() {
    // plain Java code
}

public T assertThat() {
    // plain Java code
}
```

## Fluent Interface

How?

A network of logically interconnected classes with method chaining that enables us to write (very) readable and elegant code

Why?

## Java 8 Streams

```
double avg = Arrays.asList(1,2,5,7).stream()
        .filter(...)
         .summaryStatistics()
        .getAverage();
```

## jOOQ (SQL)

```
create.select(...)
      .from(AUTHOR)
      .join(BOOK).on(...)
      .where(...)
      .and(...)
      .orderBy(...)
      .limit(2);
```

## AssertJ

```
assertThat(fruitList)
          .hasSize(9)
          .contains(“apple”,“pear”)
          .doesNotContain(“banana”);
```

Specialized and fluent:

- REST Assured
- Java 8 Streams
- jOOQ
- AssertJ

## Method Chaining

**A technique based on a special syntax for invoking multiple method calls in a single statement without storing intermediate results.**

## Person

```java
public void setName(String s) {
    // …
}

public void setAge(int age) {
    // …
}
```

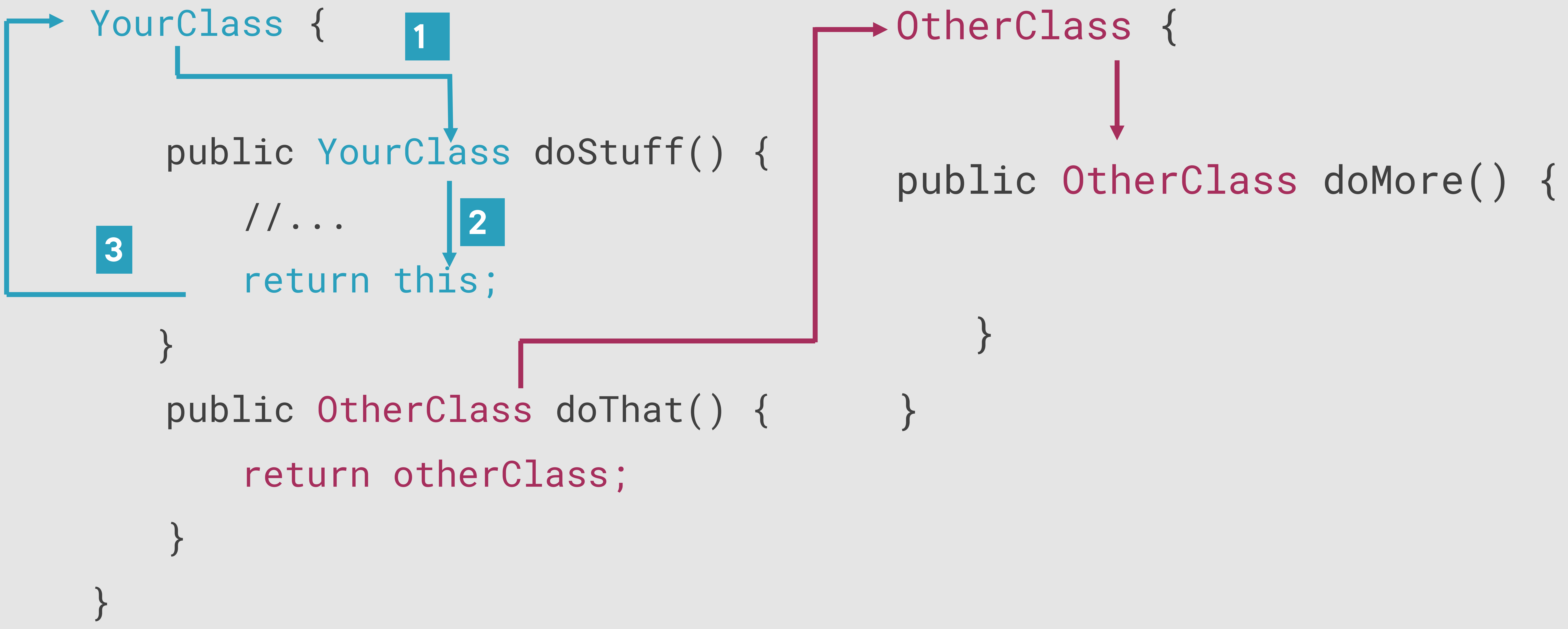
```java
Person p = new Person();


p.setName(“John”);
p.setAge(30);
```

## Person

```
public Person setName(String s) {
    // …
    return this;
}

public Person setAge(int age) {
    // …
    return this;
}
```

```
Person p =
    new Person()
    .setName("John")
    .setAge(30);
```

# Method Chaining



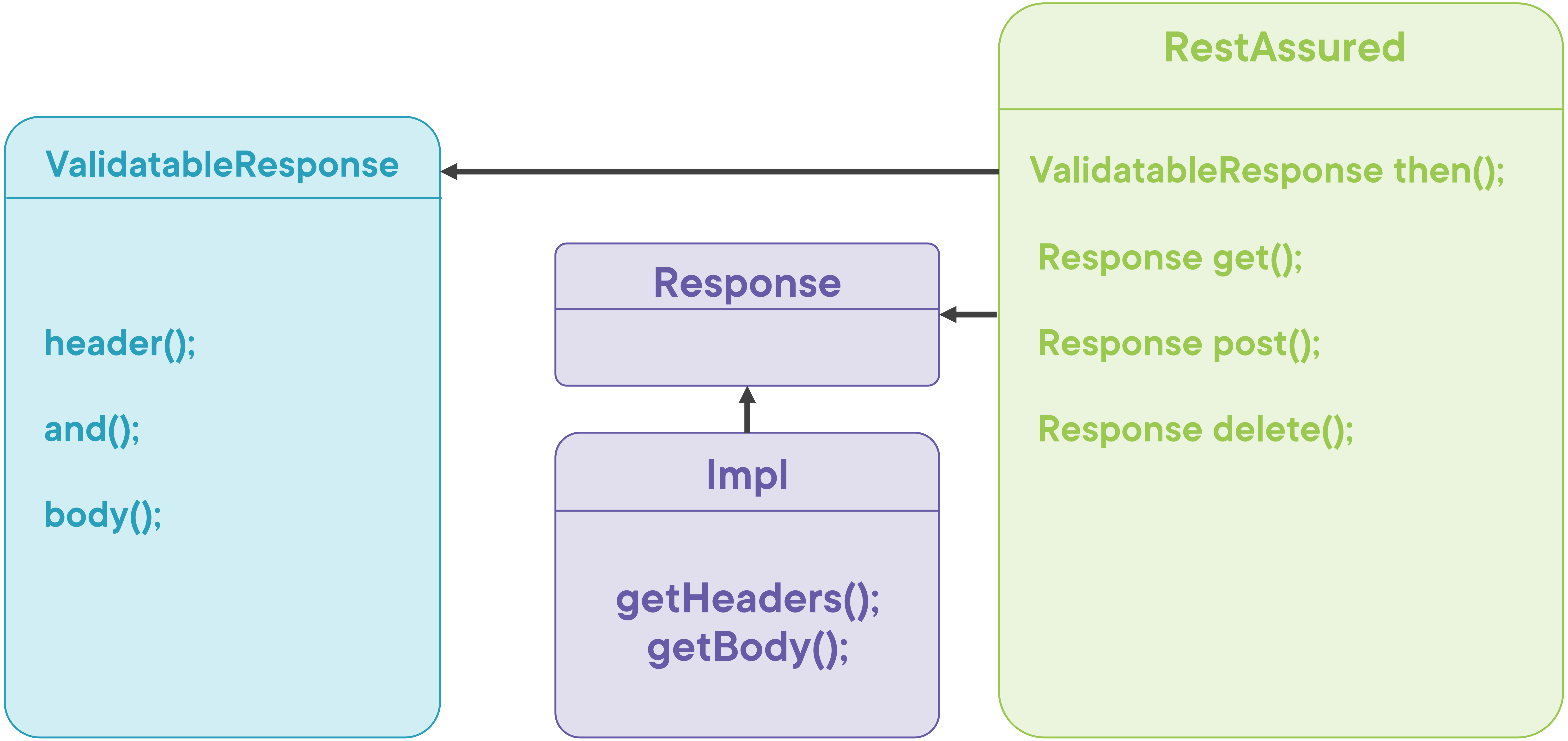
```
YourClass {

    public YourClass doStuff() {
        //...
        return this;
    }
    public OtherClass doThat() {
        return otherClass;
    }
}
```

```
OtherClass {

    public OtherClass doMore() {

    }
}
```

## Nearly finished

## Syntactic sugar

```
for (int i = 0; i < list.size(); i++) {
     // ...
}

for (int i : list) {
   // ...
}
```


ValidatableResponse
header();
and();
body();
Response
Impl
getHeaders();
getBody();
RestAssured
ValidatableResponse then();
Response get();
Response post();
Response delete();

```java
Response response = RestAssured.get("url");
String actual = response.getHeader("headerX");
assertEquals(actual,"expected");
```

## Summary

DSL with a Fluent Interface – focused and readable

Method chaining

Syntactic sugar

REST Assured primary classes