JAX-RS ハンズオン 第2部

JavaDo #08

環境セットアップ

- 教科書のURL

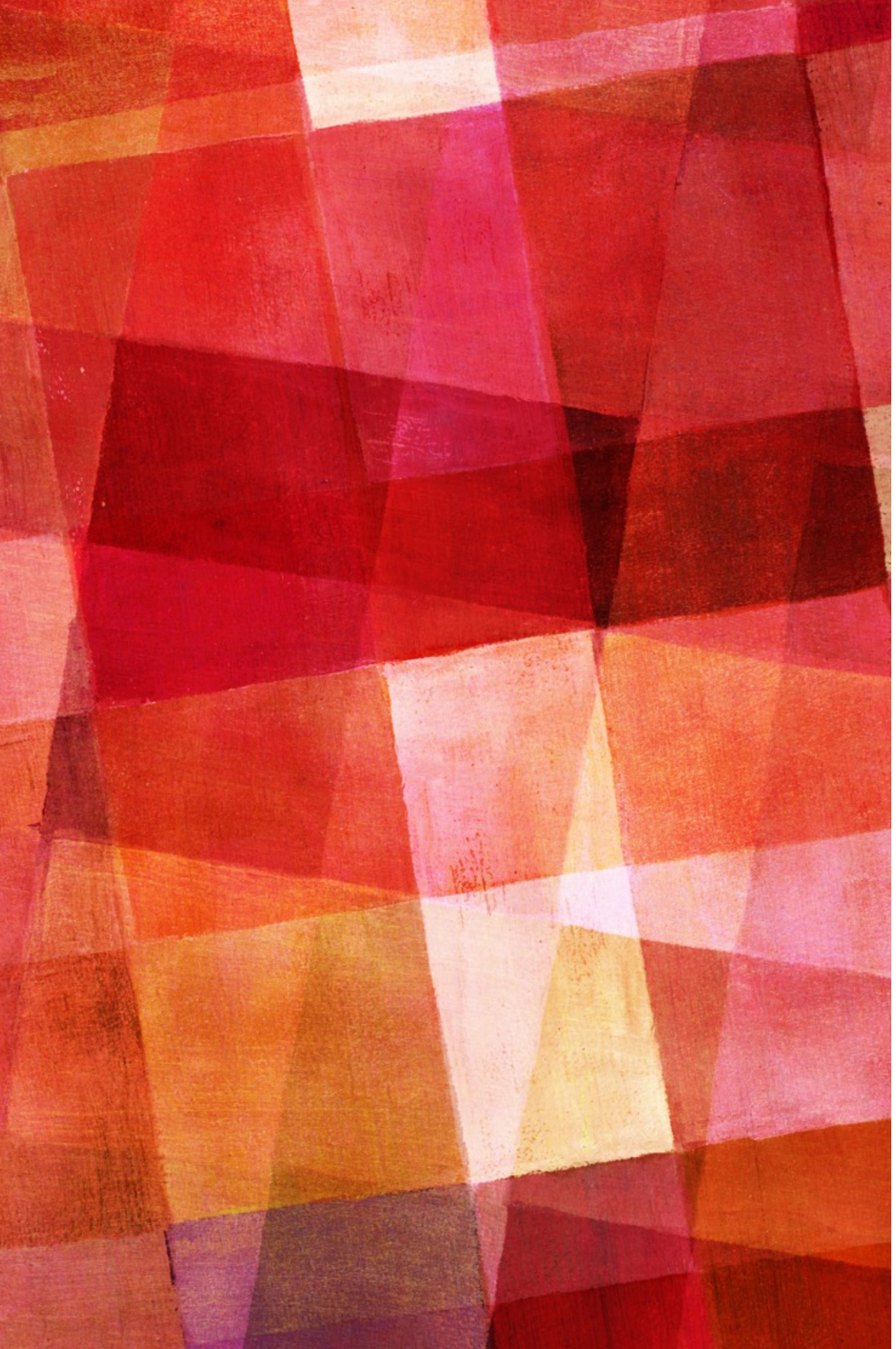
  https://goo.gl/ZeHR28

- ハンズオン[環境セットアップ]の手順URL

  https://goo.gl/m3XIAj

- 下の手順ぐらいまで順次進めてください

  1. https://github.com/java-do/20170115_seminar　にブラウザでアクセスしよう
  2. 本ハンズオン用Mavenプロジェクトをダウンロードしよう
     1. zipで取り込む方は「Clone or Download」から「Download ZIP」を選択
     2. gitで取り込む方は「https://github.com/java-do/20170115_seminar.git」で
     3. 本ハンズオンでは1の場合を前提に進めます
  3. ダウンロードしたMavenプロジェクトをIDEで読み込もう
  4. 起動の確認をしよう
  5. Advanced REST client (chromeのplugin)をインストールしよう
     - 前回入れた方は再度インストール不要です



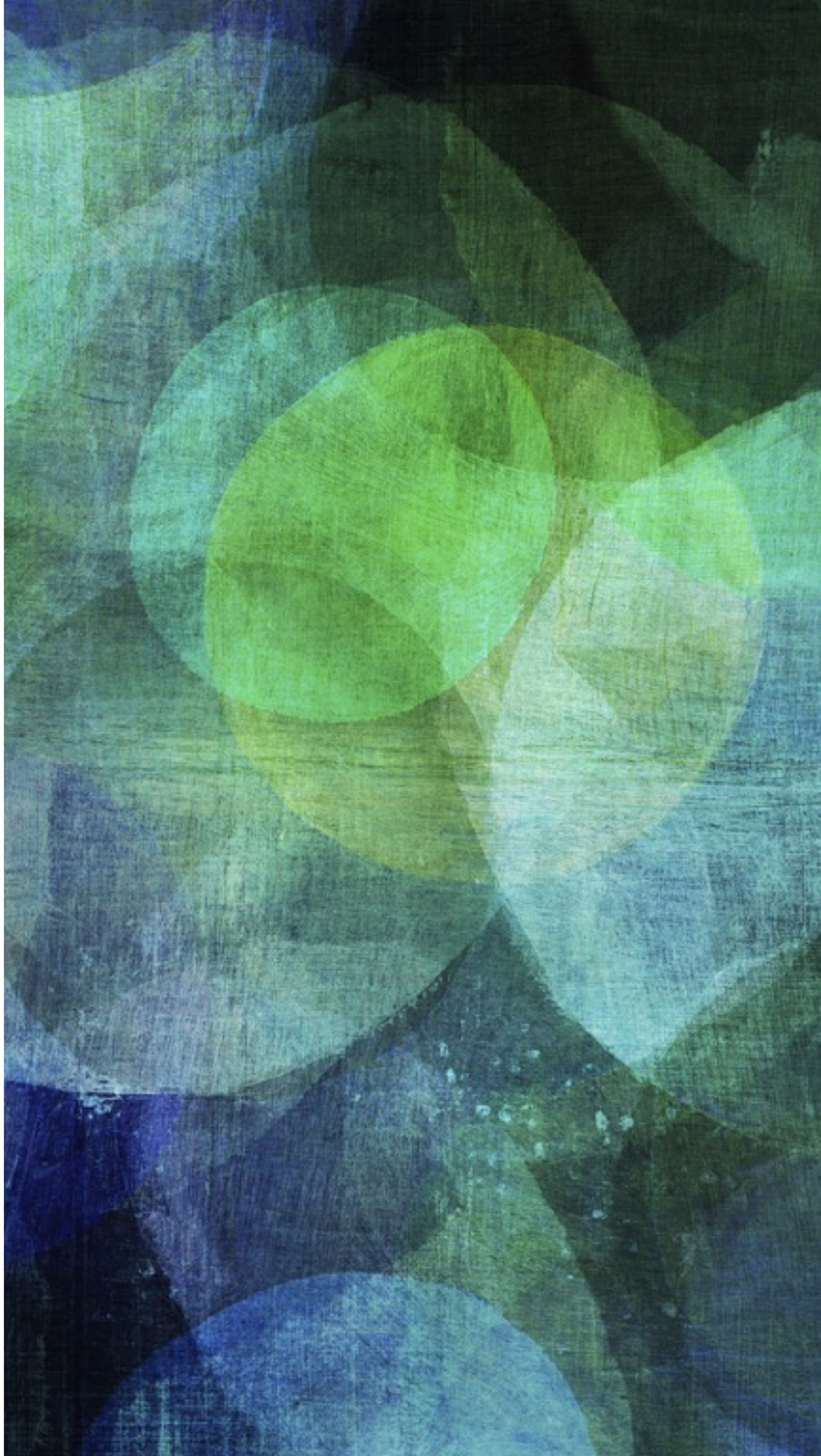
agenda

- 環境セットアップ
- 前回の復習
- 例外ハンドリング
  - ExceptionMapper
  - JSONを返す
- IPアドレスのハンドリング
- ファイルアップロード
  - Multipart版
  - シンプル版

自己紹介

- 上野 春毅
- JavaDo運営メンバー
- 燃料：コーヒー
- 最近、GuiceとSql2oのプログラム書いてます

前回の復習



RESTとは

- **Representational State Transfer(REST)**
- **Webのメディアを扱うための概念**
- HTTPかつJSON(or XML)のインターフェース
- WEB-API実装に利用

  シングルページアプリケーションなどの

  フロントエンドに対するバックエンド側

  たとえば下のようなもの

  https://goo.gl/ZkmWls

- RESTを実装するには、JavaではJAX-RS

JAX-RS

- Java API for RESTful Web Services
  - RESTに基づいたWebサービスをJavaで実装するための（Java EEの）API仕様
- 実装はアノテーションベース
- とにかくシンプルに開発できる
- JAX-RSを実装したライブラリとして、Jerseyを使います

RESTの概念

- Webでアクセスできるリソースを操作

- Webでアクセスできるとは
  - HTTPのURI（例：http://hoge.jp/ap/resource）

- リソースとは
  - 例えば、商品、アカウントなど
    - /ap/product/200
      - 200番の商品

- 操作とは
  - 取得、登録、変更、削除
    - GET、POST、PUT、DELETE

- これをHTTP通信にすると「GET /ap/resource HTTP/1.1」
  - 訳すと「アプリケーションのリソースを取得する」(HTTPバージョン1.1のプロトコルに従って)

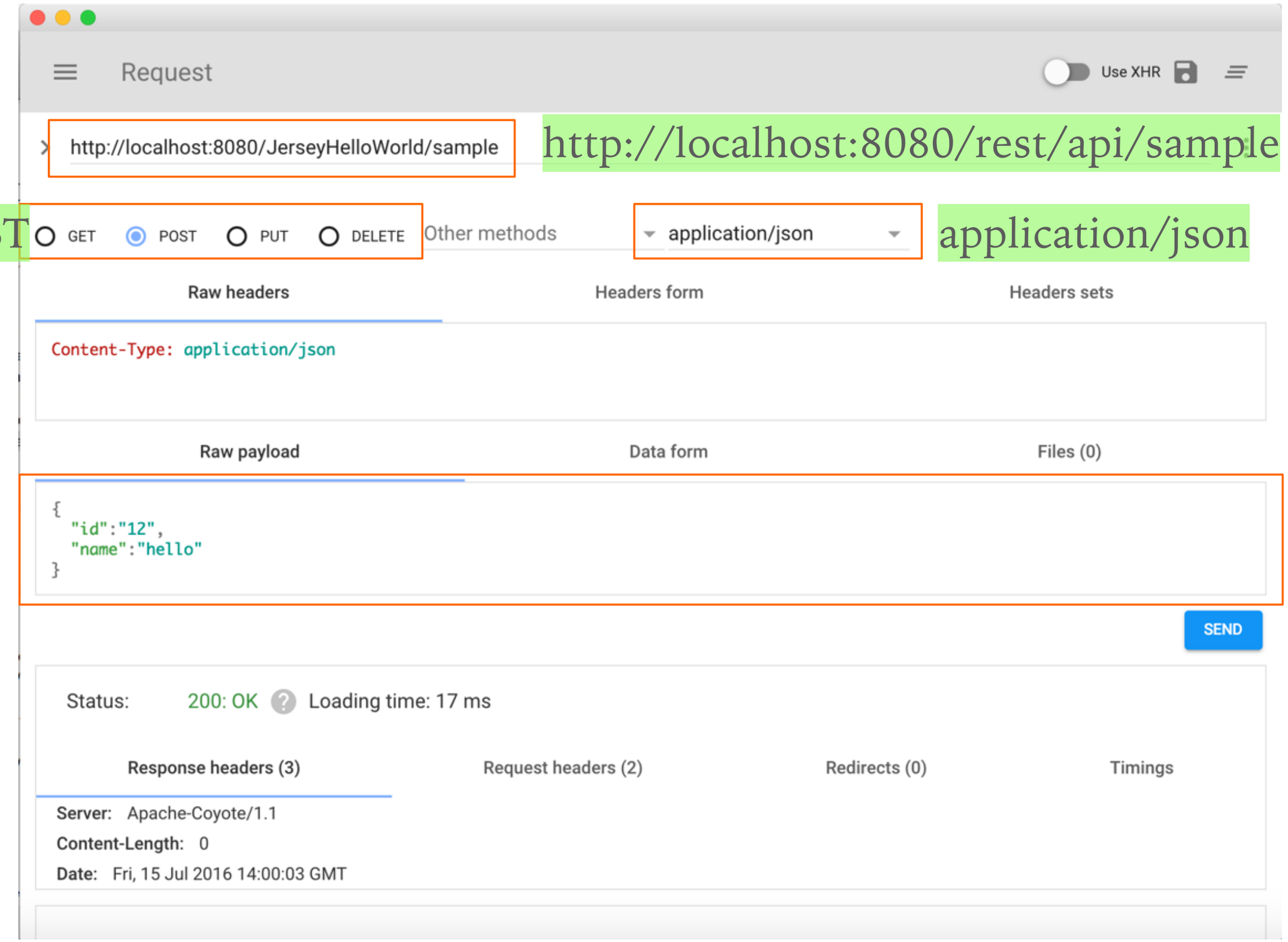
JAX-RSの実装例

```
@Path(“/product”)                    ・・・・・①URI: /ap/product
public class ProductResource {

    @GET                             ・・・・・②GETで
    @Produces(MediaType.APPLICATION_JSON)      ・・・③JSONで返却
    public Product getProduct(){                ・・・④商品を
            Product product = new Product();
            product.setName(“JavaDo“);
        return product;
    }

}
```

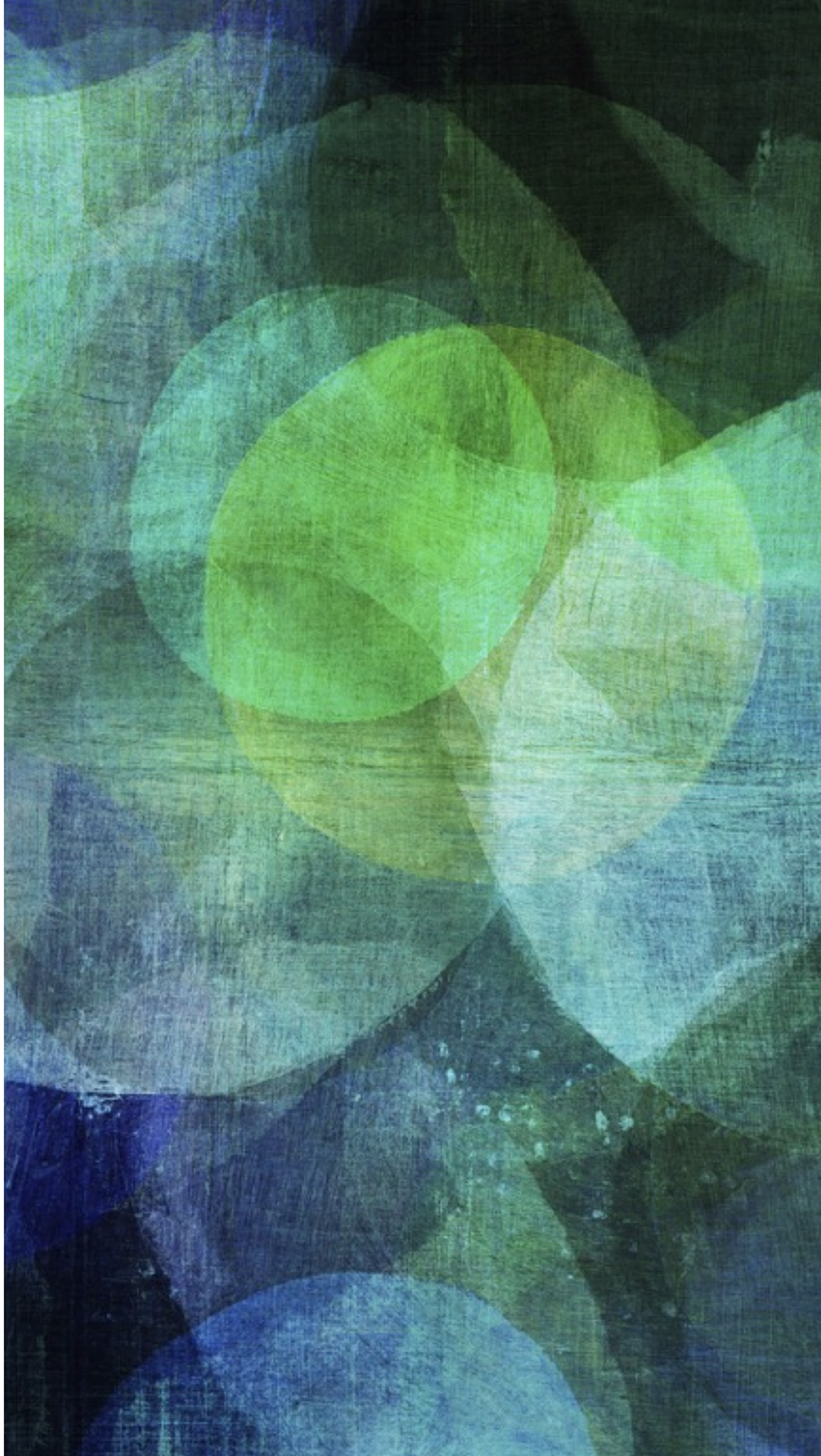
復習用にコードを書いてみましょう

- jp.javado.jaxrs.resource.SampleResourceにコードを追加しましょう
  - jp.javado.jaxrs.resource_example.SampleResourceを参照
  - @GET, @POST, @PUT, @DELETE
- 書いたらRest Clientを使って動かしてみましょう

REST Clientで叩いてみましょう



エラー処理
を作ってみましょう



エラー処理の方法 その１(前回)

- 処理の中でそのままエラー処理を書く

```
@PUT
@Path("{id}")
@Consumes(MediaType.APPLICATION_JSON)
public Response putProducts(@PathParam("id") int id,  Product product) {
  IDAOMock dao = DAOMock.getInstance();
  try {
    dao.update(id, product);
    return Response.ok().build();
  } catch (Exception e) {
    e.printStackTrace();
    int status = 400;
    return Response.status(status).build();
  }
}
```

HTTP ステータスコード:400 (Bad Request)を返却

エラー処理の方法 その２ (前回)

- WebApplicationExceptionクラスをスローする

```
@PUT
@Path("{id}")
@Consumes(MediaType.APPLICATION_JSON)
public Response putProducts(@PathParam("id") int id,  Product product) {
  IDAOMock dao = DAOMock.getInstance();
  try {
    dao.update(id, product);
    return Response.ok().build();
  } catch (Exception e) {
    e.printStackTrace();
    throw new WebApplicationException(400);
 }
}
```

エラー処理の方法 その３(前回)

- ExceptionMapperを使う場合

```
@PUT
@Path("{id}")
@Consumes(MediaType.APPLICATION_JSON)
public Response putProducts(@PathParam("id") int id,  Product product) {
  IDAOMock dao = DAOMock.getInstance();
  try {
    dao.update(id, product);
    return Response.ok().build();
  } catch (Exception e) {
    e.printStackTrace();
    throw e;
  }
}
```

この場合、HogeExceptionをスローする

エラー処理の方法　その３(今回)

- ExceptionMapperの作り方

```
@Provider
public class RuntimeExceptionMapper implements ExceptionMapper<RuntimeException> {

    @Override
    public Response toResponse(RuntimeException e) {
        System.out.println(“RuntimeExceptionMapper execute");
        e.printStackTrace();
        return Response.status(Response.Status.SERVICE_UNAVAILABLE).build();
    }
}
```

ExceptionMapper<RuntimeException>と書くと、
RuntimeExceptionが発生した場合の処理を作ることができる
ジェネリクスで指定するExceptionクラスはなんでも良い

- このクラスを書いてみましょう

エラー処理の方法　その３(今回)

*jp.javado.jaxrs.resource.ProductResource*を新規作成し、以下の内容を記述

```
import jp.javado.db.ProductDatabase;
import jp.javado.jaxrs.pojo.Product;
/* 省略 */
@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Product get(@PathParam("id") int id) throws RestException
{
    Product product = ProductDatabase.select(id);
    return product;
}
```

エラー処理の方法　その３(今回)

- ProductResourceのgetメソッド内でRuntimeExceptionをスローしてみましょう
  - まずは簡単に以下で

```
if (true) throw new RuntimeException();
```

- 書けた方は、REST Clientでアクセスしてみましょう
  - 下部のStatusが503 Service Unavailable と出る
    - Response.Status.*SERVICE_UNAVAILABLE*を指定しているため
      - Response.Status.*INTERNAL_SERVER_ERROR*とかに変えて試してみましょう

独自定義のExceptionをハンドリング

- アプリケーションで定義したExceptionクラスをハンドリングする方法についてです
- 本ハンズオン用に以下のExceptionを用意しました
  - RestException
  - RestRuntimeException

独自定義Exceptionをハンドリング

- 以下を書いてみましょう
  - RestExceptionMapperを書いてみましょう
  - RestRuntimeExceptionMapperを書いてみましょう
- 場所：jp.javado.jaxrs.exceptionパッケージに*Mapperを作成
- 今回は以下のクラスを用意しています
  - ErrorCaseはエラーの内容を表すクラス
  - ErrorMessageクラスはクライアントに返却するJson用クラス

独自定義Exceptionをハンドリング

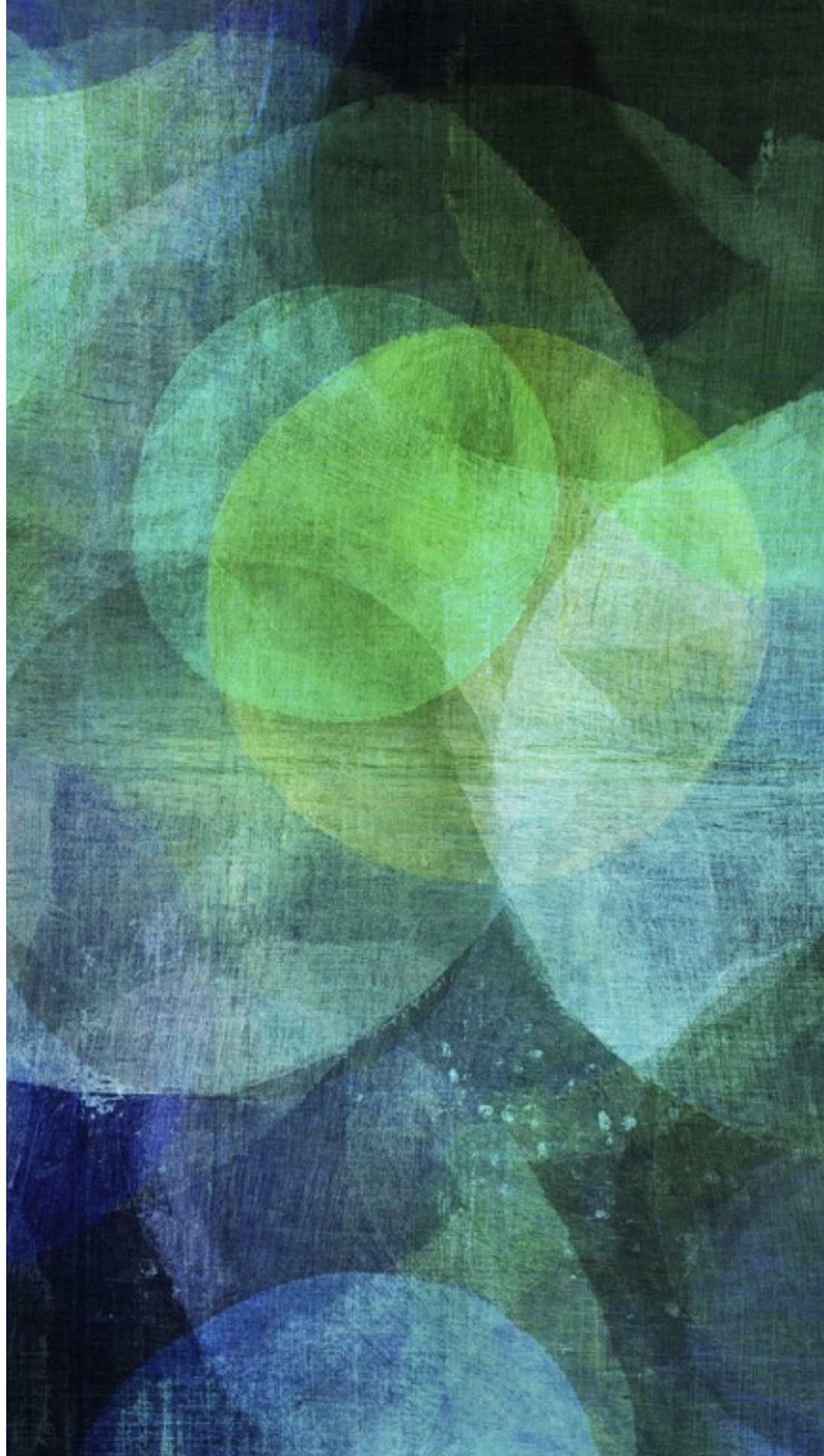
- 以下を書いてみましょう
  - RestExceptionをProductResourceに書いてみましょう
    - jp.javado.jaxrs.resourceexample.ExampleProductResource#get()を見て修正
  - RestRuntimeExceptionをProductResourceに書いてみましょう
    - 上記をRestExceptionをRestRuntimeExceptionに変更してみる

- 場所：jp.javado.jaxrs.resourceパッケージのProductResourceを加筆

独自定義Exceptionをハンドリング

- Rest Clientを使って動かしてみましょう
- アクセス方法は今までと同じ

ファイル
アップロード



ファイルアップロード

- HTTP通信のリクエストボディにファイルデータを載せてくる
  - 代表例：multipart/form-data

```
<form action="ファイル処理URI"
          method="post" enctype="multipart/form-data">
  ...
  氏名　　　：<input type="text" name="username" />
  送信ファイル：<input type="file" name="submitfile" />
  ...
</form>
```

ファイルアップロードを書いてみましょう

- jp.javado.jaxrs.resourcesample.ExampleProductResource#fileupload()
- jp.javado.jaxrs.resourcesample.ExampleProductResourceのフィールド
- 上記を書き写してみましょう

ファイルアップロードをしてみましょう

- Rest Clientでファイルアップロードの方法
    - http://localhost:8080/rest/api/product/fileupload/multipart
    - POST
    - multipart/form-data
    - [Files]を押す
        - [ADD ANOTHER FILE]を押す
        - [CHOOSE FILES]を押してファイルを選択
        - fileUploadと書いてあるところはそのまま
- [SEND]を押す
- 「Staus: 201 Created」が出たらOK、FILE_SAVE_PATHにファイルを確認

ファイルアップロードの仕組み

➤ HTTP通信で以下のリクエストが来る（通信内容全文）

```
POST /rest/api/product/fileupload/multipart HTTP/1.1
HOST: localhost:8080
content-type: multipart/form-data;
boundary=----WebKitFormBoundary4XSJradEgR8nMn8B
content-length: 914

------WebKitFormBoundary4XSJradEgR8nMn8B
Content-Disposition: form-data; name="fileUpload"; filename="sample1.jpeg"
Content-Type: image/jpg

[jpegデータの中身]


------WebKitFormBoundary4XSJradEgR8nMn8B--
```

ファイルアップロードの仕組み

➤ プログラムとの対応

```
POST /rest/api/product/fileupload/multipart HTTP/1.1
HOST: localhost:8080
content-type: multipart/form-data;
boundary=----WebKitFormBoundary4XSJradEgR8nMn8B
content-length: 914

------WebKitFormBoundary4XSJradEgR8nMn8B
Content-Disposition: form-data; name="fileUpload"; filename="sample1.jpeg"
Content-Type: image/jpg

[jpegデータの中身]

------WebKitFormBoundary4XSJradEgR8nMn8B--
```

@Consumes(MediaType.MULTIPART_FORM_DATA)

@FormDataParam("fileUpload") FormDataContentDisposition
@FormDataParam("fileUpload") FormDataBodyPart

@FormDataParam("fileUpload") InputStream

Google Drive APIで提供されているファイルアップロード

- 参照URLは以下
- https://developers.google.com/drive/v2/reference/files/insert
- 「google drive api insert」でググって一番上に出てくるのでもいけます

Media - Simple Uploadの仕様をコードで書いてみましょう

- ついでにファイル名(独自拡張)も受け取れるようにしてみましょう
- 以下を書きうつしてください
- jp.javado.jaxrs.resourceexample.ExmpleProductResource#fileuploadSimple

- 注釈（念のため！）
  - リクエストのAPI仕様を受け取ることはできる
  - レスポンス及び内部処理は仕様とは異なる
  - 認証ヘッダとかガン無視

ファイルアップロードをしてみましょう