Clear illustration can be found in : https://www.bezkoder.com/spring-boot-restcontrolleradvice/

Exception handling in springboot:

1. @RestControllerAdvice: it is used to handle exception in Restful API . it is specialization of @Comoponent annotation so it is auto-detected via class path scanning .

Rest Controller Advice’s methods (annotated with @ExceptionHandler) are shared globally across multiple @Controller components to capture exceptions and translate them to HTTP responses. The @ExceptionHandler annotation indicates which type of Exception we want to handle. The exception instance and the request will be injected via method arguments.

* @RestControler = @Controller + @ResponseBody
* @RestControllerAdvice = @ControllerAdvice + @ResponseBody

By using two annotations together, we can:

* control the body of the response along with status code
* handle several exceptions in the same method

How about @ResponseStatus?

@RestControllerAdvice annotation tells a controller that the object returned is automatically serialized into JSON and passed it to the HttpResponse object. You only need to return Java body object instead of ResponseEntity object. But the status could be always OK (200) although the data corresponds to exception signal (404 – Not Found for example). @ResponseStatus can help to set the HTTP status code for the response:

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public ErrorMessage resourceNotFoundException(ResourceNotFoundException ex, WebRequest request) {

// ...

return message;

1. @ControllerAdvice
2. @ExcepitonHandler

This approach will help to reduce the boilerplate code in springboot . When we use it then we don’t need to use time and again try and catch with in every method which might throws Internal\_SERVER\_ERROR OR DATA\_NOT\_FOUND EXCEPTION .

Meaning we can handle exception globally . we can define our own custome exception and can use that custome exception whenever we need to use within springboot application.

**Flow of Exception handle in spring boot :**

1st: make custom class which extends Exception( we can make multiple class which handle different type of exception in springboot) eg.

Public class custoemerNotFound extends Exception{

Private static final long serialversionUID-1L;

Public customerNotFound(){ }

Public customerNotFound( String errors){ }

}

2nd : make a class which Will define error response structure (Simple pojo class)

Eg

Public class Errormessage {

Private String errorCode;

Private String message;

Getter and setter

}

3rd: make ApplicationErrorHandler extends REsponseEntityExceptionHandler with annotation @RestControllerAdvice , @ExceptionHandler(ExceptionHandled.java) , @ResponseBody,@ResponseStatus in method we make in this class (this class make different method and uses those custom builte exception handler class accordingly in different method in applicaitonErrroHandler class .

Eg.

@RestControllerAdvice

Public class ExceptionControllerAdvice{

@Exceptionalhandler(CustomerNotFound.class)

Public ResponseEntity<ErrorMessage> exceptinHandler1(CustomerNotFound cnf){

ErrorMessage em = new ErrorMessage();

Em.setErrorCode(HttpStatus.BAD\_Request.value());

Em.setMessage(em.getMessage());

Return new ResponseEntity<>(em,HttpsStatus.OK);

}

@Exceptionalhandler(AddressNotFound.class)

Public ResponseEntity<ErrorMessage> exceptinHandler2(AddressNotFound cnf){

ErrorMessage em = new ErrorMessage();

Em.setErrorCode(HttpStatus.BAD\_Request.value());

Em.setMessage(em.getMessage());

Return new ResponseEntity<>(em,HttpsStatus.OK);

}

}

Note : like this we can make multiple method and can use it in controller class in method level exception throws

4th: now we use this excetion class in our RestController or @Controller class method where we can throws them in method level /.eg

@getMapping(“/customer”)

Public Customer getCustomerdetail(String id) throws CustomerNotFound , AddressNotfound{

Return new Customer();

}

@getMapping(“/Address”)

Public Address getAddress(String customerID) throws AddressNotFound {

Return new Address();

}

(Note: These all custom exceptions must be thrown in controller class in method level)