Global exception handling with @ControllerAdvice

```java
package com.penzgtu.spring.dogs.web;

import com.penzgtu.spring.dogs.model.DogDto;
import com.penzgtu.spring.dogs.repo.Dog;
import com.penzgtu.spring.dogs.service.DogsService;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("/dogs")
@RequiredArgsConstructor
@Setter
public class DogsController {
    @Autowired private final DogsService service;

    @GetMapping
    public ResponseEntity<List<Dog>>; getDogs() {
        List<Dog> dogs;

        try {
            dogs = service.getDogs();
        } catch (DogsServiceException ex) {
            return new ResponseEntity<>(null, null, HttpStatus.INTERNAL_SERVER_ERROR);
        } catch (DogsNotFoundException ex) {
            return new ResponseEntity<>(null, null, HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(dogs, HttpStatus.OK);
    }
}
```

The ***DogsController*** returns a ***ResponseEntity*** instance, which has a response body along with ***HttpStatus***.

- If no exception is thrown, the following endpoint returns ***List<Dog>*** as response body and 200 as status.
- for ***DogsNotFoundException***, it returns empty body and status 404.
- for ***DogsServiceException***, it returns 500 and empty body.

The problem with this approach is ***DUPLICATION***. The catch blocks are generic and will be needed in other endpoints as well (e.g. DELETE, POST, etc).

```java
package com.penzgtu.spring.dogs.web;

import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import static org.springframework.http.HttpStatus.INTERNAL_SERVER_ERROR;
import static org.springframework.http.HttpStatus.NOT_FOUND;

@ControllerAdvice
@Slf4j
public class DogsServiceErrorAdvice {

    @ExceptionHandler({RuntimeException.class})
    public ResponseEntity<String> handleRunTimeException(RuntimeException e) {
        return error(INTERNAL_SERVER_ERROR, e);
    }

    @ExceptionHandler({DogsNotFoundException.class})
    public ResponseEntity<String> handleNotFoundException(DogsNotFoundException e) {
        return error(NOT_FOUND, e);
    }

    @ExceptionHandler({DogsServiceException.class})
    public ResponseEntity<String> handleDogsServiceException(DogsServiceException e){
        return error(INTERNAL_SERVER_ERROR, e);
    }

    private ResponseEntity<String> error(HttpStatus status, Exception e) {
        log.error("Exception : ", e);
        return ResponseEntity.status(status).body(e.getMessage());
    }
}
```

Controller Advice *(@ControllerAdvice)*

Spring provides a better way of handling exceptions, which is Controller Advice. This is a centralized place to handle all the application level exceptions.

See what is happening here:

- *handleRunTimeException*: This method handles all the *RuntimeException* and returns the status of INTERNAL_SERVER_ERROR.
- *handleNotFoundException*: This method handles *DogsNotFoundException* and returns NOT_FOUND.
- *handleDogsServiceException*: This method handles *DogsServiceException* and returns INTERNAL_SERVER_ERROR.

The key is to catch the checked exceptions in the application and throw RuntimeExceptions. Let these exceptions be thrown out of the Controller class, and then, Spring applies the ControllerAdvice to it.

```
try {
        //
        // Lines of code
        //
} catch (SQLException sqle) {
        throw new DogsServiceException(sqle.getMessage());
}
```

Use *@ResponseStatus* to Map the Exception to the ResponseStatus

```java
package com.penzgtu.spring.dogs.web;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
public class DogsServiceErrorAdvice {

    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ExceptionHandler({DogsNotFoundException.class})
    public void handle(DogsNotFoundException e) {}

    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler({DogsServiceException.class,
SQLException.class, NullPointerException.class})
    public void handle() {}

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler({DogsServiceValidationException.class})
    public void handle(DogsServiceValidationException e) {}
}
```

Use *@ResponseStatus* With a Custom Exception

Spring also does an abstraction for the *@ControllerAdvice*, and we can even skip writing one.
The trick is to define your own RunTimeException and annotate it with a specific *@ResponseStatus*. When the particular exception is thrown out of a Controller, the Spring abstraction returns the specific Response Status.
Here is a custom RunTimeException class.

```
package com.penzgtu.spring.dogs.service;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class DogsNotFoundException extends RuntimeException {
    public DogsNotFoundException(String message) {
        super(message);
    }
}
```

Let’s throw it from anywhere in the code. For example, I am throwing it from a Service method here:

```java
public List<Dog> getDogs() {
        throw new DogsNotFoundException("No Dog Found Here..");
}
```

When you made a call to the respective Controller endpoint, you will receive 404 with the following body.

```json
{
    "timestamp": "2018-11-28T05:06:28.460+0000",
    "status": 404,
    "error": "Not Found",
    "message": "No Dog Found Here..",
    "path": "/dogs"
}
```