

Why to do exception handling?

Why to do exception handling?

In Cgbank application, consider the following REST controller :

```
1. @RestController
2. @RequestMapping(value="/cgbank")
3. public class CustomerAPI {
4.
5.     @Autowired
6.     private CustomerService customerService;
7.
8.     @GetMapping(value = "/customers/{customerId}")
9.     public ResponseEntity<Customer> getCustomerDetails(@PathVariable Integer
customerId) throws Exception {
10.         Customer customer = customerService.getCustomer(customerId);
11.         ResponseEntity<Customer> response = new
ResponseEntity<Customer>(customer, HttpStatus.OK);
12.         return response;
13.     }
14. }
```

This controller has cgbank/customer/{customerId} mapping which takes customerId and returns the customer details if customerId is present.

If no customer is present with given customer id and Service class throws an exception with message Service.CUSTOMER_UNAVAILABLE for non existent customer then this controller returns the following response:

```
1. {
2.     "timestamp": "2020-06-18T08:20:20.705+0000",
3.     "status": 500,
4.     "error": "Internal Server Error",
5.     "trace": "com.infy.exception.CgbankException:
Service.CUSTOMER_NOT_FOUND",
6.     "message": "Service.CUSTOMER_UNAVAILABLE",
7.     "path": "/cgbank/customers/8"
8. }
```

In above response, the HTTP status code is 500 which conveys that there is some internal server error which is incorrect. Since the resource is not present the status code should be 404 which means resource not found. So, we can modify the controller code as shown below:

```
1. @RestController
2. @RequestMapping(value="/cgbank")
3. public class CustomerAPI {
4.
5.     @Autowired
6.     private CustomerService customerService;
7.
8.     @GetMapping(value = "/customers/{customerId}")
9.     public ResponseEntity<Customer> getCustomerDetails(@PathVariable Integer
customerId) throws Exception {
10.         ResponseEntity<Customer> response = null;
11.         try {
12.             Customer customer =
customerService.getCustomer(customerId);
```

```

13.         response = new ResponseEntity<Customer>(customer,
14.           HttpStatus.OK);
15.     } catch (Exception exception) {
16.         return new ResponseEntity<>(HttpStatus.NOT_FOUND);
17.     }
18.     return response;
19. }

```

But this solution adds more code to the controller and makes it complex. Also if multiple handler methods are present in controller, then this code you may have to repeat in all those methods. Also, there is no way to send more detailed description of exception to the client. So proper exception handling in REST API's is very important. Spring provides multiple ways of exception handling in REST API. In this course, you will learn following two ways of exception handling:

1. Using `@RestControllerAdvice`
2. Using `ResponseStatusException` class

Exception Handling using `@RestControllerAdvice`

In this approach, a central exception handler class is created to handle all the unhandled exceptions thrown from your code. So all the unhandled exceptions will be handled at one central place which makes exception handling easier. Now let us see how to do this.

To do this the first create a central exception handler class and annotate it with it **`@RestControllerAdvice`** annotation as shown below:

```

1. @RestControllerAdvice
2. public class ExceptionControllerAdvice{
3.     // exception handler methods
4. }
5.

```

Then add handler methods to this class to handle different types of exceptions and annotate them with **`@ExceptionHandler`** annotation.

Now to handle `CgbankException` exception, add exception handler method as follows:

```

1. @RestControllerAdvice
2. public class ExceptionControllerAdvice {
3.
4.     @Autowired
5.     private Environment environment;
6.
7.     @ExceptionHandler(CgbankException.class)
8.     public ResponseEntity<String>
9.     cgbankExceptionHandler(CgbankException exception){
10.         return new
11.         ResponseEntity<>(environment.getProperty(exception.getMes
12.         sage()), HttpStatus.NOT_FOUND);
13.     }
14. }

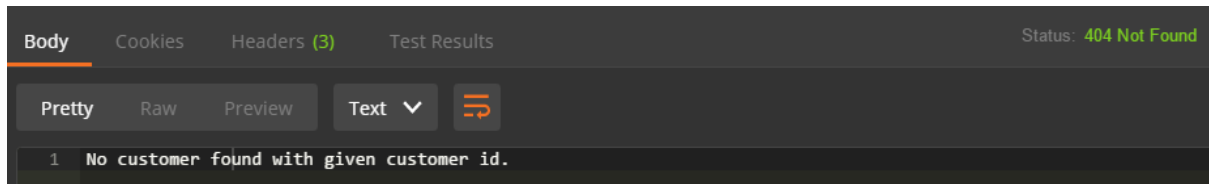
```

```

10.         }
11.     }
12.

```

Now if your code throws `CgbankException` the `cgbankExceptionHandler()` method will execute. Now if you to fetch details of non existent customer you will get following response with correct HTTP status code from controller:



You can also have multiple handler methods in the Advice class to handle exceptions of different types and return the response accordingly. For example, to handle all types of exception you can also add a general exception handler method as follows:

```

1. @RestControllerAdvice
2. public class ExceptionControllerAdvice {
3.
4.     @Autowired
5.     private Environment environment;
6.
7.     @ExceptionHandler(Exception.class)
8.     public ResponseEntity<String> exceptionHandler(Exception
exception) {
9.         return new
ResponseEntity<>(environment.getProperty("General.EXCEPTI
ON_MESSAGE"), HttpStatus.INTERNAL_SERVER_ERROR);
10.     }
11.
12.     @ExceptionHandler(CgbankException.class)
13.     public ResponseEntity<String>
cgbankExceptionHandler(CgbankException exception){
14.         return new
ResponseEntity<>(environment.getProperty(exception.getMes
sage()), HttpStatus.NOT_FOUND);
15.     }
16. }
17.

```

If you want one method to handle both `Exception` and `CgbankException`, then you can write one handler method as follows:

```

1. @RestControllerAdvice
2. public class RestExceptionHandler {
3.     @ExceptionHandler( { Exception.class,
CgbankException.class })

```

```

4.         public ResponseEntity<String> exceptionHandler(Exception
exception){
5.             // rest of the code
6.         }
7.     }

```

You have seen that when exception is thrown the response from the controller only contains message and the HTTP status code. But sometimes you want to provide more information to the client about the exception. For example, you want to provide details about date and time of exception also along with message and HTTP status code. To do this you have to create a custom error class with desired information as follows:

```

1. public class ErrorInfo {
2.     private String errorMessage;
3.     private Integer errorCode;
4.     private LocalDateTime timestamp;
5.     //getters and setters
6. }

```

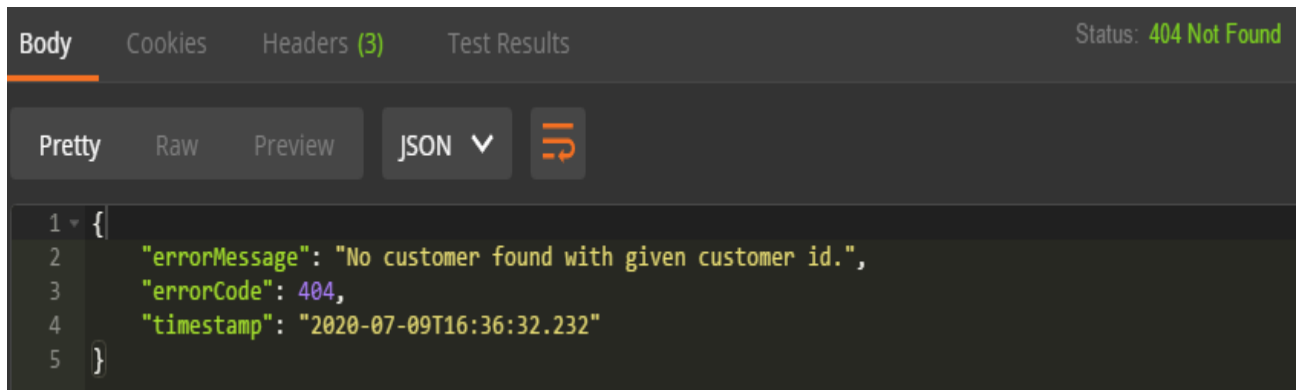
Now, modify the handler methods of RestExceptionHandler class to include object of ErrorInfo class as part of response as shown below:

```

1. @RestControllerAdvice
2. public class ExceptionControllerAdvice {
3.     @Autowired
4.     Environment environment;
5.
6.     @ExceptionHandler(Exception.class)
7.     public ResponseEntity<ErrorInfo> exceptionHandler(Exception exception) {
8.         ErrorInfo error = new ErrorInfo();
9.
10.        error.setErrorMessage(environment.getProperty("General.EXCEPTION_MESSAGE"));
11.        ;
12.        error.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
13.        error.setTimestamp(LocalDateTime.now());
14.        return new ResponseEntity<ErrorInfo>(error,
15.        HttpStatus.INTERNAL_SERVER_ERROR);
16.    }
17.
18.    @ExceptionHandler(CgbankException.class)
19.    public ResponseEntity<ErrorInfo> cgbankExceptionHandler(CgbankException
20.    exception) {
21.        ErrorInfo error = new ErrorInfo();
22.
23.        error.setErrorMessage(environment.getProperty(exception.getMessage()));
24.        error.setTimestamp(LocalDateTime.now());
25.        error.setErrorCode(HttpStatus.NOT_FOUND.value());
26.        return new ResponseEntity<ErrorInfo>(error, HttpStatus.NOT_FOUND);
27.    }
28. }

```

. Now if you fetch details of non existent customer you will get following response from controller:



```
1 {  
2   "errorMessage": "No customer found with given customer id.",  
3   "errorCode": 404,  
4   "timestamp": "2020-07-09T16:36:32.232"  
5 }
```

Exception Handling using @RestControllerAdvice - Demo

Objective:

To illustrate how to do global exception handling in Spring REST application

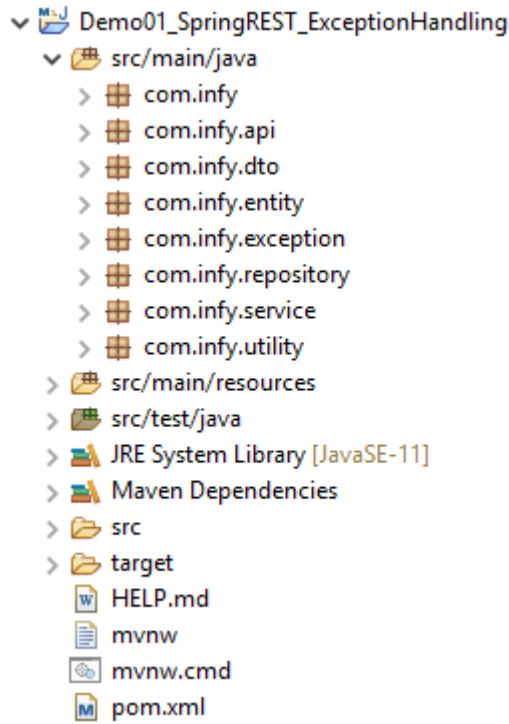
Steps:

Step 1: Using Spring Initializr, create a Spring Boot project with following specifications:

- Spring Boot Version: 2.3.1
- Group: com.infy
- Artifact: Demo01_SpringREST_ExceptionHandling
- Name: Demo01_SpringREST_ExceptionHandling
- Package name: com.infy
- Java Version: 11
- Dependencies: Spring Data JPA, MySQL Driver, Spring Web, Spring Boot DevTools.

Now, import this project in Eclipse.

Step 2: Modify the imported project according to the following project structure :



Step 3: Create the database and table

Open MySQL terminal and execute the following command:

```
1. drop database if exists customer_db;
2. create database customer_db;
3. use customer_db;
4. create table customer(
5.     customer_id int auto_increment,
6.     email_id varchar(50),
7.     name varchar(20),
8.     date_of_birth date,
9.     constraint ps_customer_id_pk primary key (customer_id)
10.);
11. insert into customer (customer_id, email_id, name, date_of_birth) values (1,
12.     'martin@infy.com', 'Martin', sysdate()- interval 9000 day);
13. insert into customer (customer_id, email_id, name, date_of_birth) values (2,
14.     'tim@infy.com', 'Tim', sysdate()- interval 5000 day);
15. insert into customer (customer_id, email_id, name, date_of_birth) values (3,
16.     'jack@infy.com', 'Jack', sysdate()- interval 6000 day);
17. commit;
18. select * from customer;
```

Step 4: Configure the data source

Open application.properties in src/main/resources folder and add following properties for MySQL:

```
1. #MySQL settings
2. #Change these settings according to database you are using
3. spring.datasource.url=jdbc:mysql://localhost:3306/customer_db
4. spring.datasource.username=root
5. #If MySQL installation is password proctored, then use below property to set
   password
6. spring.datasource.password=root
7. #JPA settings
8. spring.jpa.show-sql=true
```

```
9. spring.jpa.properties.hibernate.format_sql=true
```

Step 5: Create the following CustomerDTO class in com.infy.dto package :

```
1. public class CustomerDTO {
2.
3.     private Integer customerId;
4.     private String emailId;
5.     private String name;
6.     private LocalDate dateOfBirth;
7.
8.     public Integer getCustomerId() {
9.         return customerId;
10.    }
11.
12.    public void setCustomerId(Integer customerId) {
13.        this.customerId = customerId;
14.    }
15.
16.    public String getEmailId() {
17.        return emailId;
18.    }
19.
20.    public void setEmailId(String emailId) {
21.        this.emailId = emailId;
22.    }
23.
24.    public String getName() {
25.        return name;
26.    }
27.
28.    public void setName(String name) {
29.        this.name = name;
30.    }
31.
32.    public LocalDate getDateOfBirth() {
33.        return dateOfBirth;
34.    }
35.
36.    public void setDateOfBirth(LocalDate dateOfBirth) {
37.        this.dateOfBirth = dateOfBirth;
38.    }
39.
40.    @Override
41.    public String toString() {
42.        return "Customer [customerId=" + customerId + ", emailId=" +
43.        emailId + ", name=" + name + ", dateOfBirth="
44.        + dateOfBirth + " ]";
45.    }
46. }
```

Step 6: Create the following Customer class in com.infy.entity package :

```
1. @Entity
2. public class Customer {
3.
4.     @Id
5.     @GeneratedValue(strategy=GenerationType.IDENTITY)
6.     private Integer customerId;
7.     private String emailId;
8.     private String name;
9.     private LocalDate dateOfBirth;
10. }
```

```

11. public Integer getCustomerId() {
12.     return customerId;
13. }
14.
15. public void setCustomerId(Integer customerId) {
16.     this.customerId = customerId;
17. }
18.
19. public String getEmailId() {
20.     return emailId;
21. }
22.
23. public void setEmailId(String emailId) {
24.     this.emailId = emailId;
25. }
26.
27. public String getName() {
28.     return name;
29. }
30.
31. public void setName(String name) {
32.     this.name = name;
33. }
34.
35. public LocalDate getDateOfBirth() {
36.     return dateOfBirth;
37. }
38.
39. public void setDateOfBirth(LocalDate dateOfBirth) {
40.     this.dateOfBirth = dateOfBirth;
41. }
42.
43. @Override
44. public int hashCode() {
45.     return 31;
46. }
47.
48. @Override
49. public String toString() {
50.     return "Customer [customerId=" + customerId + ", emailId=" +
emailId + ", name=" + name + ", dateOfBirth="
51.         + dateOfBirth + "]\n";
52. }
53.
54. @Override
55. public boolean equals(Object obj) {
56.     if (this == obj)
57.         return true;
58.     if (obj == null)
59.         return false;
60.     if (this.getClass() != obj.getClass())
61.         return false;
62.     Customer other = (Customer) obj;
63.     if (this.getCustomerId() == null) {
64.         if (other.getCustomerId() != null)
65.             return false;
66.     } else if (!this.getCustomerId().equals(other.getCustomerId()))
67.         return false;
68.     return true;
69. }
70.
71.
72. }
73.

```

Step 7: Create the following CustomerRepository interface in com.infy.repository package :


```

1. public interface CustomerRepository extends CrudRepository<Customer,
   Integer> {
2.
3. }
4.

```

Step 8: Create the following CustomerService interface in com.infy.service package :

```

1. public interface CustomerService {
2.     public Integer addCustomer(CustomerDTO customerDTO) throws CgbankException;
3.     public CustomerDTO getCustomer(Integer customerId) throws CgbankException;
4.     public void updateCustomer(Integer customerId, String emailId) throws
   CgbankException;
5.     public void deleteCustomer(Integer customerId) throws CgbankException;
6.     public List<CustomerDTO> getAllCustomers() throws CgbankException;
7. }

```

Step 9: Create the following CustomerServiceImpl class in com.infy.service package :

```

1. @Service(value = "customerService")
2. @Transactional
3. public class CustomerServiceImpl implements CustomerService {
4.
5.     @Autowired
6.     private CustomerRepository customerRepository;
7.
8.     @Override
9.     public CustomerDTO getCustomer(Integer customerId) throws CgbankException {
10.         Optional<Customer> optional =
   customerRepository.findById(customerId);
11.         Customer customer = optional.orElseThrow(() -> new
   CgbankException("Service.CUSTOMER_NOT_FOUND"));
12.         CustomerDTO customer2 = new CustomerDTO();
13.         customer2.setCustomerId(customer.getCustomerId());
14.         customer2.setDateOfBirth(customer.getDateOfBirth());
15.         customer2.setEmailId(customer.getEmailId());
16.         customer2.setName(customer.getName());
17.         return customer2;
18.     }
19.
20.     @Override
21.     public Integer addCustomer(CustomerDTO customerDTO) throws CgbankException
   {
22.         Customer customerEntity = new Customer();
23.         customerEntity.setDateOfBirth(customerDTO.getDateOfBirth());
24.         customerEntity.setEmailId(customerDTO.getEmailId());
25.         customerEntity.setName(customerDTO.getName());
26.         customerEntity.setCustomerId(customerDTO.getCustomerId());
27.         Customer customerEntity2 =
   customerRepository.save(customerEntity);
28.         return customerEntity2.getCustomerId();
29.     }
30.
31.     @Override
32.     public void updateCustomer(Integer customerId, String emailId) throws
   CgbankException {
33.         Optional<Customer> customer =
   customerRepository.findById(customerId);
34.         Customer c = customer.orElseThrow(() -> new
   CgbankException("Service.CUSTOMER_NOT_FOUND"));
35.         c.setEmailId(emailId);
36.     }
37.

```

```

38.  @Override
39.  public void deleteCustomer(Integer customerId) throws CgbankException {
40.      Optional<Customer> customer =
customerRepository.findById(customerId);
41.      customer.orElseThrow(() -> new
CgbankException("Service.CUSTOMER NOT FOUND"));
42.      customerRepository.deleteById(customerId);
43.  }
44.
45.  @Override
46.  public List<CustomerDTO> getAllCustomers() throws CgbankException {
47.      Iterable<Customer> customers = customerRepository.findAll();
48.      List<CustomerDTO> customerDTOs = new ArrayList<>();
49.      customers.forEach(customer -> {
50.          CustomerDTO cust = new CustomerDTO();
51.          cust.setCustomerId(customer.getCustomerId());
52.          cust.setDateOfBirth(customer.getDateOfBirth());
53.          cust.setEmailId(customer.getEmailId());
54.          cust.setName(customer.getName());
55.          customerDTOs.add(cust);
56.      });
57.      if (customerDTOs.isEmpty())
58.          throw new CgbankException("Service.CUSTOMERS_NOT_FOUND");
59.      return customerDTOs;
60.  }
61.
62. }

```

Step 10: Create an aspect LoggingAspect class in com.infy.utility package:

```

1.  @Component
2.  @Aspect
3.  public class LoggingAspect {
4.      public static final Log LOGGER = LoggerFactory.getLog(LoggingAspect.class);
5.      @AfterThrowing(pointcut = "execution(* com.infy.service.*Impl.*(..))",
throwing = "exception")
6.      public void logServiceException(Exception exception) {
7.          LOGGER.error(exception.getMessage(), exception);
8.      }
9.  }

```

Step 11: Create the following CgbankException class in com.infy.exception package :

```

1.  public class CgbankException extends Exception {
2.
3.      private static final long serialVersionUID = 1L;
4.
5.      public CgbankException(String message) {
6.          super(message);
7.      }
8.  }
9.

```

Step 12: Create the following ErrorInfo class in com.infy.utility package :

```

1.  public class ErrorInfo {
2.      private String errorMessage;
3.      private Integer errorCode;
4.      private LocalDateTime timestamp;
5.
6.      public String getErrorMessage() {

```

```

7.         return errorMessage;
8.     }
9.
10.    public void setErrorMessage(String errorMessage) {
11.        this.errorMessage = errorMessage;
12.    }
13.
14.    public Integer getErrorCode() {
15.        return errorCode;
16.    }
17.
18.    public void setErrorCode(Integer errorCode) {
19.        this.errorCode = errorCode;
20.    }
21.
22.    public LocalDateTime getTimestamp() {
23.        return timestamp;
24.    }
25.
26.    public void setTimestamp(LocalDateTime timestamp) {
27.        this.timestamp = timestamp;
28.    }
29.
30. }
31.

```

Step 13: Create the following ExceptionControllerAdvice class in com.infy.utility package :

```

1. @RestControllerAdvice
2. public class ExceptionControllerAdvice {
3.     @Autowired
4.     Environment environment;
5.
6.     @ExceptionHandler(Exception.class)
7.     public ResponseEntity<ErrorInfo> exceptionHandler(Exception exception) {
8.         ErrorInfo error = new ErrorInfo();
9.
10.        error.setErrorMessage(environment.getProperty("General.EXCEPTION_MESSAGE"));
11.        ;
12.        error.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
13.        error.setTimestamp(LocalDateTime.now());
14.        return new ResponseEntity<ErrorInfo>(error,
15.        HttpStatus.INTERNAL_SERVER_ERROR);
16.    }
17.
18.    @ExceptionHandler(CgbankException.class)
19.    public ResponseEntity<ErrorInfo> cgbankExceptionHandler(CgbankException
20.    exception) {
21.        ErrorInfo error = new ErrorInfo();
22.
23.        error.setErrorMessage(environment.getProperty(exception.getMessage()));
24.        error.setTimestamp(LocalDateTime.now());
25.        error.setErrorCode(HttpStatus.NOT_FOUND.value());
26.        return new ResponseEntity<ErrorInfo>(error, HttpStatus.NOT_FOUND);
27.    }
28.
29. }
30.
31.

```

Step 14: Create the following CustomerAPI class in com.infy.api package :

```

1. @RestController
2. @RequestMapping(value = "/cgbank")
3. public class CustomerAPI {

```

```

4.
5.     @Autowired
6.     private CustomerService customerService;
7.
8.     @Autowired
9.     private Environment environment;
10.
11.     @GetMapping(value = "/customers")
12.     public ResponseEntity<List<CustomerDTO>> getAllCustomers() throws
CgbankException {
13.         List<CustomerDTO> customerDTOs =
customerService.getAllCustomers();
14.         return new ResponseEntity<>(customerDTOs, HttpStatus.OK);
15.     }
16.
17.     @GetMapping(value = "/customers/{customerId}")
18.     public ResponseEntity<CustomerDTO> getCustomer(@PathVariable Integer
customerId) throws CgbankException {
19.         CustomerDTO customerDTO = customerService.getCustomer(customerId);
20.         return new ResponseEntity<>(customerDTO, HttpStatus.OK);
21.     }
22.
23.     @PostMapping(value = "/customers")
24.     public ResponseEntity<String> addCustomer(@RequestBody CustomerDTO
customerDTO) throws CgbankException {
25.         Integer customerId = customerService.addCustomer(customerDTO);
26.         String successMessage =
environment.getProperty("API.INSERT_SUCCESS") + customerId;
27.         return new ResponseEntity<>(successMessage, HttpStatus.CREATED);
28.     }
29.
30.     @PutMapping(value = "/customers/{customerId}")
31.     public ResponseEntity<String> updateCustomer(@PathVariable Integer
customerId, @RequestBody CustomerDTO customer)
32.         throws CgbankException {
33.         customerService.updateCustomer(customerId, customer.getEmailId());
34.         String successMessage =
environment.getProperty("API.UPDATE_SUCCESS");
35.         return new ResponseEntity<>(successMessage, HttpStatus.OK);
36.     }
37.
38.     @DeleteMapping(value = "/customers/{customerId}")
39.     public ResponseEntity<String> deleteCustomer(@PathVariable Integer
customerId) throws CgbankException {
40.         customerService.deleteCustomer(customerId);
41.         String successMessage =
environment.getProperty("API.DELETE_SUCCESS");
42.         return new ResponseEntity<>(successMessage, HttpStatus.OK);
43.     }
44. }

```

Step 15: Add the following properties in application.properties:

```

1. Service.CUSTOMER_NOT_FOUND=No customer found with given customer id.
2. Service.CUSTOMERS_NOT_FOUND=No customers found.
3. General.EXCEPTION_MESSAGE=Request could not be processed due to some issue.
   Please try again!
4.
5. API.INSERT_SUCCESS=Customer added successfully with customer id :
6. API.UPDATE_SUCCESS=Customer emailid successfully updated.
7. API.DELETE_SUCCESS=Customer details deleted successfully.
8.
9. server.port=8765
10.

```

Step 16: The log4j2.properties files should look like this and it should be placed in resources folder.

```
1. #Name of the Properties file
2. name=LoggerConfigFile
3.
4. #Declaring logger for business logic
5. logger.file.name=com.infy.utility
6. logger.file.level=DEBUG
7. logger.file.appenderRef.file.ref=LoggerAppender
8. logger.file.additivity=false
9.
10. #Declaring logger for business console
11. logger.console.name=com.infy
12. logger.console.level=INFO
13. logger.console.appenderRef.file.ref=ConsoleAppender
14. logger.console.additivity=false
15.
16. # File Appender
17. appender.file.name=LoggerAppender
18. appender.file.type=File
19. appender.file.fileName=log/ErrorLog.log
20. #Logging Pattern
21. appender.file.layout.type=PatternLayout
22. appender.file.layout.pattern=%d{dd-MMM-yyyy HH:mm:ss} %level - %m%n
23.
24.
25. # Console Appender
26. appender.console.name=ConsoleAppender
27. appender.console.type=Console
28. #Logging Pattern for console
29. appender.console.layout.type=PatternLayout
30. appender.console.layout.pattern=%m%n
31.
32.
```

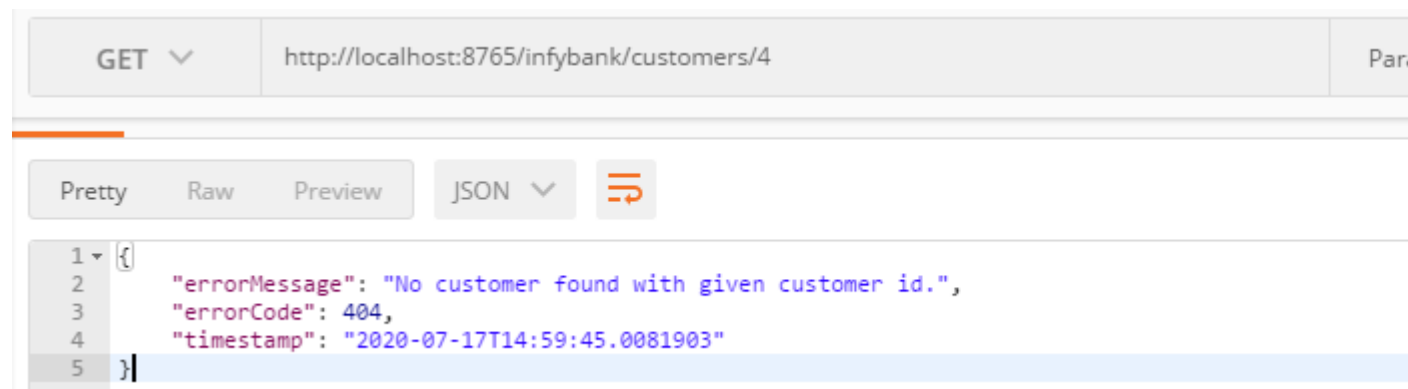
Step 17: Deploy the application on the server by executing the class containing the main method.

We have successfully implemented global exception handling. Now, you can test it using Postman client.

Testing Web Service using Postman

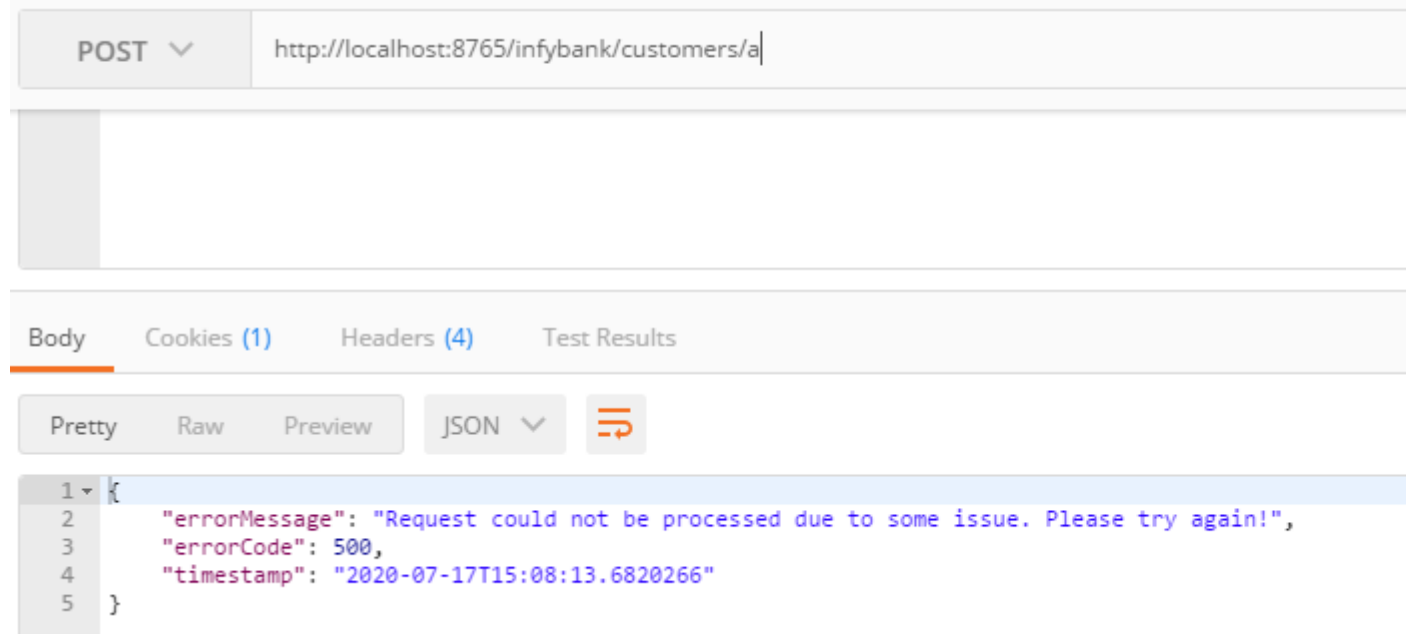
Step 1 : Launch Postman.

Step 2 : Provide HTTP GET request - <http://localhost:8765/cgbank/customers/4> that will get the customer details based on the customerId (invalid customerId that does not exist in cgbank).



We observe that Exception is thrown with code 404. This Exception is of type CgbankException.

Step 3 : Provide HTTP GET request - **http://localhost:8765/cgbank/customers/a** that will get the customer details based on the customerId (A character is passed instead of an integer)



We observe that Exception is thrown with code 500. This Exception is of type Exception.

We can conclude that Global Exception handling is working for exceptions of both CgbankException and Exception types.

Exception handling using ResponseStatusException

Spring 5 introduced the ResponseStatusException class. This is a base class for exceptions associated with specific HTTP status codes. You can create an instance of it by providing an HttpStatus code, a message to explain the exception and cause of the exception as follows:

```
1. @GetMapping(value = "/customers/{customerId}")
2. public ResponseEntity<CustomerDTO> getCustomer(@PathVariable Integer
   customerId) throws CgbankException {
3.     try {
4.         CustomerDTO customerDTO =
   customerService.getCustomer(customerId);
5.         return new ResponseEntity<>(customerDTO, HttpStatus.OK);
6.     } catch (Exception exception) {
7.         throw new ResponseStatusException(HttpStatus.NOT_FOUND,
   environment.getProperty(exception.getMessage()), exception);
8.     }
9. }
```

Exception handling using ResponseStatusException - Demo

Objective :

This demo illustrates how to do throw `ResponseStatusException` in a Spring REST application.

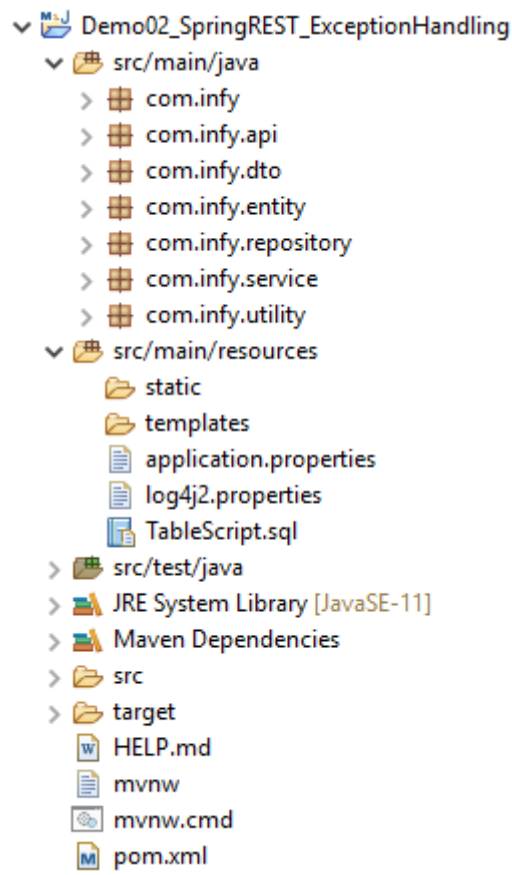
Steps :

Step 1: Using Spring Initializr, create a Spring Boot project with following specifications :

- Spring Boot Version: 2.3.1
- Group: com.infy
- Artifact: Demo02_SpringREST_ExceptionHandling
- Name: Demo02_SpringREST_ExceptionHandling
- Package name: com.infy
- Java Version: 11
- Dependencies: Spring Data JPA, MySQL Driver, Spring Web and Devtools.

Now, import this project in Eclipse.

Step 2: Modify the imported project according to the following project structure :



Step 3: Create the database and table

Open MySQL terminal and execute the following command:

```
1. drop database if exists customer_db;
2. create database customer_db;
3. use customer_db;
4. create table customer(
5.     customer_id int auto_increment,
6.     email_id varchar(50),
7.     name varchar(20),
8.     date of birth date,
```

```

9.     constraint ps_customer_id_pk primary key (customer_id)
10.);
11.insert into customer (customer_id, email_id, name, date_of_birth) values (1,
    'martin@infy.com', 'Martin', sysdate()- interval 9000 day);
12.insert into customer (customer_id, email_id, name, date_of_birth) values (2,
    'tim@infy.com', 'Tim', sysdate()- interval 5000 day);
13.insert into customer (customer_id, email_id, name, date_of_birth) values (3,
    'jack@infy.com', 'Jack', sysdate()- interval 6000 day);
14.commit;

15.select * from customer;

```

Step 4: Configure the data source

Open application.properties in src/main/resources folder and add following properties for MySQL:

```

1. #MySQL settings
2. #Change these settings according to database you are using
3. spring.datasource.url=jdbc:mysql://localhost:3306/customer_db
4. spring.datasource.username=root
5. #If MySQL installation is password proctored, then use below property to set
   password
6. spring.datasource.password=root
7. #JPA settings
8. spring.jpa.show-sql=true
9. spring.jpa.properties.hibernate.format_sql=true

```

Step 5: Create the following Customer DTO class in com.infy.dto package :

```

1. public class CustomerDTO {
2.
3.     private Integer customerId;
4.     private String emailId;
5.     private String name;
6.     private LocalDate dateOfBirth;
7.
8.     public Integer getCustomerId() {
9.         return customerId;
10.    }
11.
12.    public void setCustomerId(Integer customerId) {
13.        this.customerId = customerId;
14.    }
15.
16.    public String getEmailId() {
17.        return emailId;
18.    }
19.
20.    public void setEmailId(String emailId) {
21.        this.emailId = emailId;
22.    }
23.
24.
25.    public String getName() {
26.        return name;
27.    }
28.
29.    public void setName(String name) {
30.        this.name = name;
31.    }
32.
33.    public LocalDate getDateOfBirth() {
34.        return dateOfBirth;
35.    }

```



```

36.
37. public void setDateOfBirth(LocalDate dateOfBirth) {
38.     this.dateOfBirth = dateOfBirth;
39. }
40.
41.
42. @Override
43. public String toString() {
44.
45.     return "Customer [customerId=" + customerId + ", emailId=" +
emailId + ", name=" + name + ", dateOfBirth="
46.         + dateOfBirth + "];"
47. }
48. }
49.

```

Step 6: Create the Customer class in com.infy.entity package :

```

1. @Entity
2. public class Customer {
3.
4.     @Id
5.     @GeneratedValue(strategy=GenerationType.IDENTITY)
6.     private Integer customerId;
7.     private String emailId;
8.     private String name;
9.     private LocalDate dateOfBirth;
10.
11.
12. public Integer getCustomerId() {
13.     return customerId;
14. }
15.
16. public void setCustomerId(Integer customerId) {
17.     this.customerId = customerId;
18. }
19.
20. public String getEmailId() {
21.     return emailId;
22. }
23.
24. public void setEmailId(String emailId) {
25.     this.emailId = emailId;
26. }
27.
28. public String getName() {
29.     return name;
30. }
31.
32. public void setName(String name) {
33.     this.name = name;
34. }
35.
36. public LocalDate getDateOfBirth() {
37.     return dateOfBirth;
38. }
39.
40. public void setDateOfBirth(LocalDate dateOfBirth) {
41.     this.dateOfBirth = dateOfBirth;
42. }
43.
44. @Override
45. public int hashCode() {
46.
47.     return 31;
48. }

```

```

49.
50. @Override
51. public String toString() {
52.     return "Customer [customerId=" + customerId + ", emailId=" +
emailId + ", name=" + name + ", dateOfBirth="
53.         + dateOfBirth + "];"
54. }
55.
56. @Override
57. public boolean equals(Object obj) {
58.     if (this == obj)
59.         return true;
60.     if (obj == null)
61.         return false;
62.     if (this.getClass() != obj.getClass())
63.         return false;
64.     Customer other = (Customer ) obj;
65.     if (this.getCustomerId() == null) {
66.         if (other.getCustomerId() != null)
67.             return false;
68.     } else if (!this.getCustomerId().equals(other.getCustomerId()))
69.         return false;
70.     return true;
71. }
72.
73.
74. }
75.

```

Step 7: Create the following CustomerRepository interface in com.infy.repository package :

```

1. public interface CustomerRepository extends CrudRepository<Customer,
Integer> {
2.
3. }
4.

```

Step 8: Create the following CustomerService interface in com.infy.service package:

```

1. public interface CustomerService {
2.     public Integer addCustomer(CustomerDTO customerDTO) throws CgbankException;
3.     public CustomerDTO getCustomer(Integer customerId) throws CgbankException;
4.     public void updateCustomer(Integer customerId, String emailId) throws
CgbankException;
5.     public void deleteCustomer(Integer customerId) throws CgbankException;
6.     public List<CustomerDTO> getAllCustomers() throws CgbankException;
7. }

```

Step 9: Create the following CustomerServiceImpl class in com.infy.service package:

```

1. @Service(value = "customerService")
2. @Transactional
3. public class CustomerServiceImpl implements CustomerService {
4.
5.     @Autowired
6.     private CustomerRepository customerRepository;
7.
8.     @Override
9.     public CustomerDTO getCustomer(Integer customerId) throws CgbankException {
10.         Optional<Customer> optional =
customerRepository.findById(customerId);

```

```

11.         Customer customer = optional.orElseThrow(() -> new
CgbankException("Service.CUSTOMER_NOT_FOUND"));
12.         CustomerDTO customer2 = new CustomerDTO();
13.         customer2.setCustomerId(customer.getCustomerId());
14.         customer2.setDateOfBirth(customer.getDateOfBirth());
15.         customer2.setEmailId(customer.getEmailId());
16.         customer2.setName(customer.getName());
17.         return customer2;
18.     }
19.
20.     @Override
21.     public Integer addCustomer(CustomerDTO customerDTO) throws CgbankException
{
22.         Customer customerEntity = new Customer();
23.         customerEntity.setDateOfBirth(customerDTO.getDateOfBirth());
24.         customerEntity.setEmailId(customerDTO.getEmailId());
25.         customerEntity.setName(customerDTO.getName());
26.         customerEntity.setCustomerId(customerDTO.getCustomerId());
27.         Customer customerEntity2 =
customerRepository.save(customerEntity);
28.         return customerEntity2.getCustomerId();
29.     }
30.
31.     @Override
32.     public void updateCustomer(Integer customerId, String emailId) throws
CgbankException {
33.         Optional<Customer> customer =
customerRepository.findById(customerId);
34.         Customer c = customer.orElseThrow(() -> new
CgbankException("Service.CUSTOMER_NOT_FOUND"));
35.         c.setEmailId(emailId);
36.     }
37.
38.     @Override
39.     public void deleteCustomer(Integer customerId) throws CgbankException {
40.         Optional<Customer> customer =
customerRepository.findById(customerId);
41.         customer.orElseThrow(() -> new
CgbankException("Service.CUSTOMER_NOT_FOUND"));
42.         customerRepository.deleteById(customerId);
43.     }
44.
45.     @Override
46.     public List<CustomerDTO> getAllCustomers() throws CgbankException {
47.         Iterable<Customer> customers = customerRepository.findAll();
48.         List<CustomerDTO> customerDTOs = new ArrayList<>();
49.         customers.forEach(customer -> {
50.             CustomerDTO cust = new CustomerDTO();
51.             cust.setCustomerId(customer.getCustomerId());
52.             cust.setDateOfBirth(customer.getDateOfBirth());
53.             cust.setEmailId(customer.getEmailId());
54.             cust.setName(customer.getName());
55.             customerDTOs.add(cust);
56.         });
57.         if (customerDTOs.isEmpty())
58.             throw new CgbankException("Service.CUSTOMERS_NOT_FOUND");
59.         return customerDTOs;
60.     }
61.
62. }

```

Step 10: Create an aspect LoggingAspect class in com.infy.utility package:

```

1. @Component
2. @Aspect
3. public class LoggingAspect {

```

```

4.     public static final Log LOGGER = LoggerFactory.getLog(LoggingAspect.class);
5.     @AfterThrowing(pointcut = "execution(* com.infy.service.*Impl.*(..))",
    throwing = "exception")
6.     public void logServiceException(Exception exception) {
7.         LOGGER.error(exception.getMessage(), exception);
8.     }
9. }

```

Step 11: Create the following CustomerAPI class in com.infy.api package :

```

1. @RestController
2. @RequestMapping(value = "/cgbank")
3. public class CustomerAPI {
4.
5.     @Autowired
6.     private CustomerService customerService;
7.
8.     @Autowired
9.     private Environment environment;
10.
11.     @GetMapping(value = "/customers")
12.     public ResponseEntity<List<CustomerDTO>> getAllCustomers() throws
    CgbankException {
13.         try {
14.             List<CustomerDTO> customerDTOs =
    customerService.getAllCustomers();
15.             return new ResponseEntity<>(customerDTOs, HttpStatus.OK);
16.         } catch (Exception exception) {
17.             throw new ResponseStatusException(HttpStatus.NOT_FOUND,
    environment.getProperty(exception.getMessage()), exception);
18.         }
19.     }
20.
21.     @GetMapping(value = "/customers/{customerId}")
22.     public ResponseEntity<CustomerDTO> getCustomer(@PathVariable Integer
    customerId) throws CgbankException {
23.         try {
24.             CustomerDTO customerDTO =
    customerService.getCustomer(customerId);
25.             return new ResponseEntity<>(customerDTO, HttpStatus.OK);
26.         } catch (Exception exception) {
27.             throw new ResponseStatusException(HttpStatus.NOT_FOUND,
    environment.getProperty(exception.getMessage()), exception);
28.         }
29.     }
30.
31.     @PostMapping(value = "/customers")
32.     public ResponseEntity<String> addCustomer(@RequestBody CustomerDTO
    customerDTO) throws CgbankException {
33.         try {
34.             Integer customerId =
    customerService.addCustomer(customerDTO);
35.             String successMessage =
    environment.getProperty("API.INSERT_SUCCESS") + customerId;
36.             return new ResponseEntity<>(successMessage,
    HttpStatus.CREATED);
37.         } catch (Exception exception) {
38.             throw new ResponseStatusException(HttpStatus.NOT_FOUND,
    environment.getProperty(exception.getMessage()), exception);
39.         }
40.     }
41.
42.     @PutMapping(value = "/customers/{customerId}")
43.     public ResponseEntity<String> updateCustomer(@PathVariable Integer
    customerId, @RequestBody CustomerDTO customer)
44.         throws CgbankException {

```

```

45.         try {
46.             customerService.updateCustomer(customerId,
customer.getEmailId());
47.             String successMessage =
environment.getProperty("API.UPDATE_SUCCESS");
48.             return new ResponseEntity<>(successMessage,
HttpStatus.OK);
49.         } catch (Exception exception) {
50.             throw new ResponseStatusException(HttpStatus.NOT_FOUND,
environment.getProperty(exception.getMessage()), exception);
51.         }
52.     }
53.
54.     @DeleteMapping(value = "/customers/{customerId}")
55.     public ResponseEntity<String> deleteCustomer(@PathVariable Integer
customerId) throws CgbankException {
56.         try {
57.             customerService.deleteCustomer(customerId);
58.             String successMessage =
environment.getProperty("API.DELETE_SUCCESS");
59.             return new ResponseEntity<>(successMessage,
HttpStatus.OK);
60.         } catch (Exception exception) {
61.             throw new ResponseStatusException(HttpStatus.NOT_FOUND,
environment.getProperty(exception.getMessage()), exception);
62.         }
63.     }
64. }

```

Step 12: Add the following to application.properties.

```

1. Service.CUSTOMER_NOT_FOUND=No customer found with given customer id.
2. Service.CUSTOMERS_NOT_FOUND=No customers found.
3. General.EXCEPTION_MESSAGE=Request could not be processed due to some issue.
Please try again!
4.
5. API.INSERT_SUCCESS=Customer added successfully with customer id :
6. API.UPDATE_SUCCESS=Customer emailid successfully updated.
7. API.DELETE_SUCCESS=Customer details deleted successfully.
8.
9. server.port=8765
10.
11.
12.

```

Step 13: Deploy the application on the server by executing the class containing the main method.

We have successfully implemented Exception Handling using ResponseStatusException.

You can now test using Postman client.

Testing Web Service using Postman

Step 1: Launch Postman.

Step 2: Test this URL - <http://localhost:8765/cgbank/customers/5> using HTTP GET that will get the customer based on the customerId (invalid customerId that does not exist in cgbank) being passed.

```
GET http://localhost:8765/infybank/customers/5 Params Send Save
```

```
1 {
2   "timestamp": "2020-07-17T09:47:48.428+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "trace": "org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND \\\"No customer found with given customer id.\\\"; nested exception is",
6   "message": "No customer found with given customer id.",
7   "path": "/infybank/customers/5"
8 }
```

We observe that Exception is thrown with code 404 is thrown, in the body of the error we can see the full stack trace

Step 3 : Test this URL - <http://localhost:8765/cgbank/customers/a> using HTTP GET that will get the customer based on the customerId being passed. (A character is passed instead of an integer)

```
GET http://localhost:8765/infybank/customers/a Params Send Save
```

```
1 {
2   "timestamp": "2020-07-17T09:44:36.189+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "trace": "org.springframework.web.method.annotation.MethodArgumentTypeMismatchException: Failed to convert value of type 'java.lang.String' to req",
6   "message": "Failed to convert value of type 'java.lang.String' to required type 'java.lang.Integer'; nested exception is java.lang.NumberFormatExc",
7   "path": "/infybank/customers/a"
8 }
```

We observe that Exception is thrown with code 400 is thrown, in the body of the error we can see the full stack trace

We can conclude that Exception handling is working with `ResponseStatusException`.