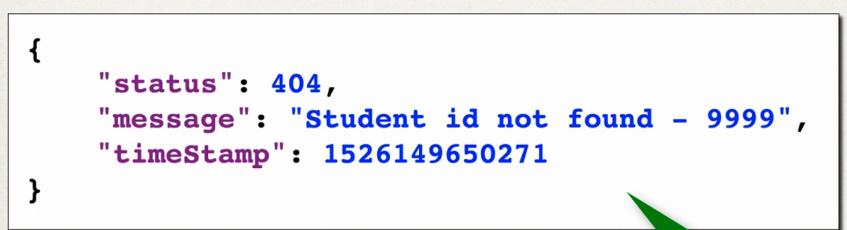
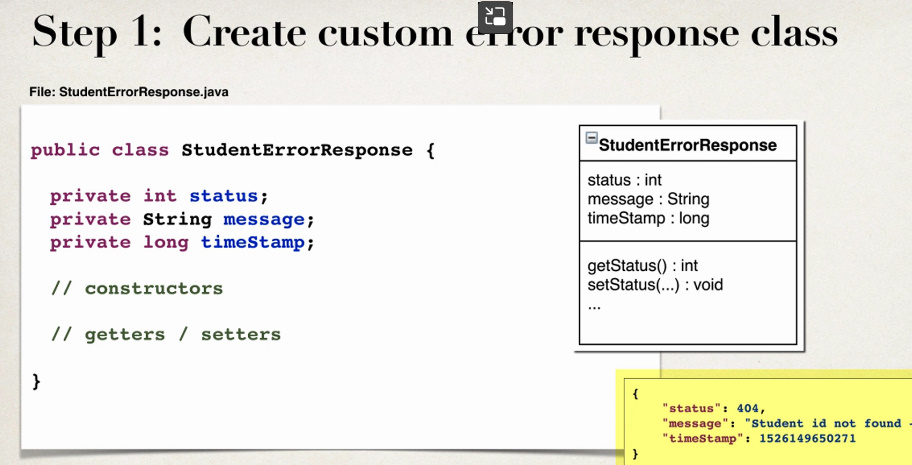
**Necesitatea**

* Atunci cand introducem un request invalid, ca .../api/students/9999, eroarea care apare e urata si arata si date despre configurarea aplicatiei noastre.
* O solutie ar fi sa returnam un JSON de eroare, ca acesta:



**Pasi**

1. Cream o clasa model care sa fie returnata ca JSON error:

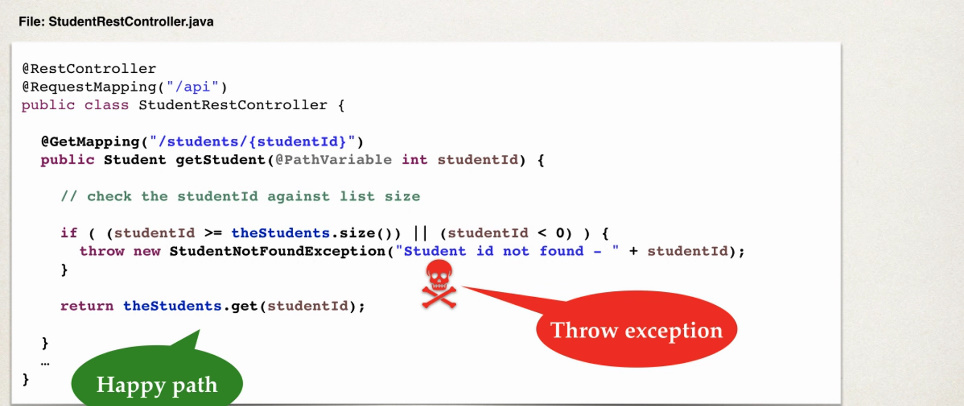


1. Cream a Custom Student Excpetion class

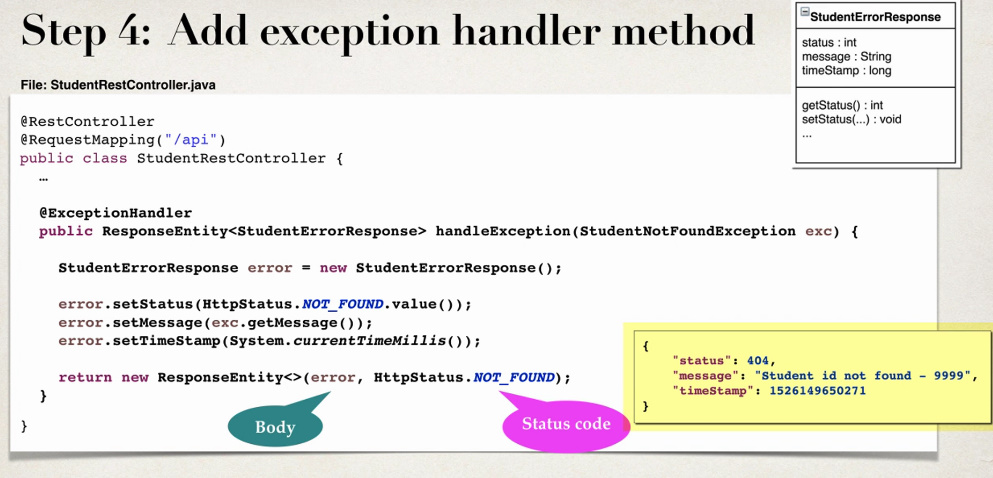
public class StudentNotFoundException extends RuntimeException{  
  
 public StudentNotFoundException(String message) {  
 super(message);  
 }  
   
}

1. Facem ca REST Service sa arunce o exceptie

@RestController  
@RequestMapping("/api")  
public class Controller {  
  
 private List<Student> students;  
  
 @PostConstruct  
 public void getStudents(){  
 students = new ArrayList<Student>();  
 students.add(new Student("Mititiuc","Eduard"));  
 students.add(new Student("Goncear","Alex"));  
 students.add(new Student("Petru","Turcan"));  
 }  
 @GetMapping("/students/{studentId}")  
 public Student getStudents(@PathVariable int studentId){  
  
 if(studentId>=students.size() || studentId<0){  
 throw new StudentNotFoundException("Student id not found - "+studentId);  
 }  
  
 return students.get(studentId);  
 }  
}



1. Cream un exception handler cu @ExceptionHandler. **Fiecare clasa Controller poate sa aiba propriu ExceptionHandler!**





**1 –** acest @ExceptionHandler va returna entitati drept raspuns la exceptie, si anume un obiect de tip ResponseEntity ce va avea in el un StudentErrorResponse, pe care l-am creat sa fie returnat in JSON. Anume StudentErrorResponse e tipul de obiect ce vrem sa fie returnat de metoda ca response si anume ResponseEntity e cea care mereu foloseste Jackson pentru a returna orice obiect in JSON. **Deci orice obiect va fi creat si returnat in JSON de catre ResponseEntity. Putem returna pur si simplu obiectul, fara a fi de tip ResponseEntity, caci in RestController avem suport de la Jackson si el oricum il va converti in JSON**

**2** – parametru primit de exceptie cand va fi aruncata. Ca si catch()

**3** – cand aruncam un obiect de tip ResponseEntity, trebuie sa adaugam in constructor neaparat obiectul ce va fi transformat in JSON si tipul errorii.

@ExceptionHandler  
public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException e){  
 StudentErrorResponse response = new StudentErrorResponse();  
 response.setMessage(e.getMessage());  
 response.setStatus(HttpStatus.*NOT\_FOUND*.value());  
 response.setTimeStamp(System.*currentTimeMillis*());  
  
 return new ResponseEntity<StudentErrorResponse>(response,HttpStatus.*NOT\_FOUND*);  
}

E posibil sa cream si asa un ExceptionHandler:

@ExceptionHandler  
public StudentError exception1(StudentException exception){  
 StudentError studentError = new StudentError();  
 studentError.setFirstName(exception.getMessage());  
 studentError.setLastName(String.*valueOf*(HttpStatus.*NOT\_FOUND*.value()));  
 studentError.setEmail(String.*valueOf*(System.*currentTimeMillis*()));  
  
 return studentError;  
}

Totusi, ResponseEntity ne ofera posibilitatea de a trimite un HTTP response complet, cu toate elementele necesare.

Putem adauga si un Handler pentru orice alta exceptie:

@ExceptionHandler  
public ResponseEntity<StudentError> general(Exception exception){  
 StudentError studentError = new StudentError();  
 studentError.setFirstName(exception.getMessage());  
 studentError.setLastName(String.*valueOf*(HttpStatus.*BAD\_REQUEST*.value()));  
 studentError.setEmail(String.*valueOf*(System.*currentTimeMillis*()));  
  
 return new ResponseEntity<StudentError>(studentError,HttpStatus.*BAD\_REQUEST*);  
}

**Despre @ExceptionHandler**

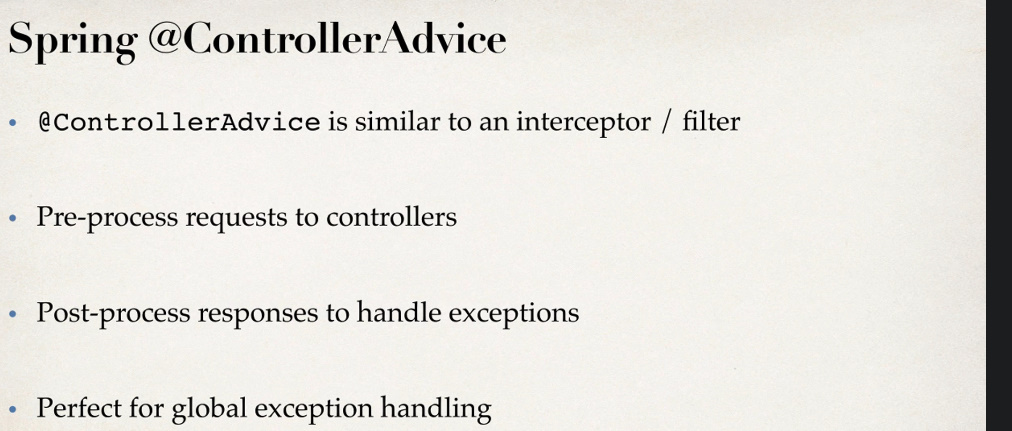
Acest @ExceptionHandler e ca un catch(), dar care returneaza ceva.

Toate exceptiile din program care nu sunt gestionate si ar trebui sa ajunga in gesiunea compilatorului, sunt preluate de @ExceptionHandler.

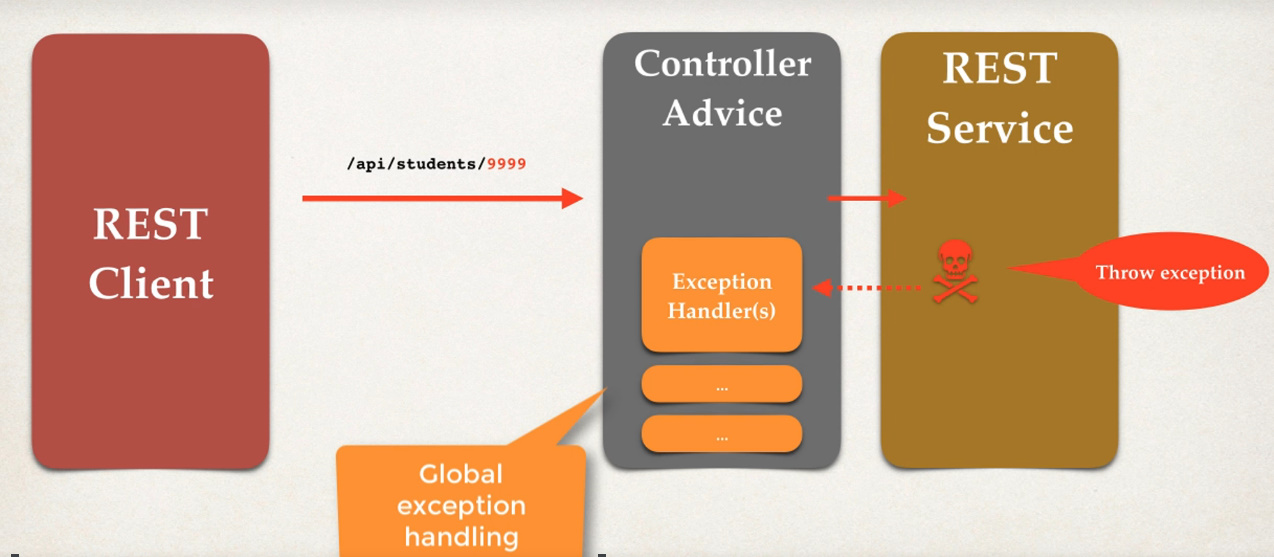
1. **El functioneaza doar in controllerul unde e definit**.
2. **Fiecare metoda trebuie sa inceapa cu @ExceptionHandler, daca ea e pentru a gestiona exceptii**.

**Global Exception Handling**

* @ExceptionHandler e doar pentru Clasa Controller in care se afla, si atat. El nu poate fi folosit de alte Controllere, si asta e o problema, caci alte parti din aplicatie nu il pot folosi nici de acum si trebuie sa rescrise mereu acelasi cod.
* Global Exception Hndling permite crearea unor exception handler globale ce pot fi reutilizate de orice controller
* Putem crea unul cu @ControllerAdvice, care este ca un filtru/interceptor
* Putem sa il folosim pentru a preprocesa requesturile la Controllers
* Putem postprocesa raspunsurile pentru a gestiona exceptiile.
* Este o utilizare a AOP, caci putem pre procesa requesturilee la controllers si post procesa raspunsuri pentru a ne ocupa de exceptii

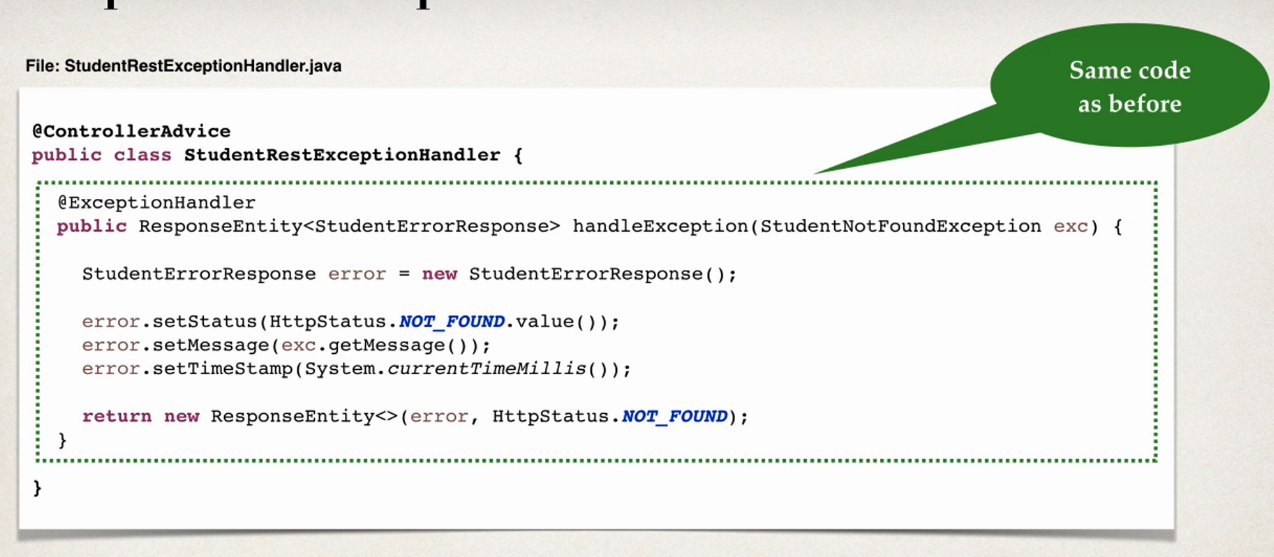


**Cum functioneaza**



Orice raspuns sau instructiune ce vine de la REST Client trece intai prin Controller Advice si abea apoi, odata ce a trecut prin el, ajunge la REST Service. Trece prin el, din cauza ca daca apare vreo Exceptie in REST Service, ea sa fie aruncata si sa nu ajunga in gestiunea programului, ci in gestiunea la Controller Advice si tot el sa returneze raspunsul la REST Client. **Toate instructiunile trec prin el**!Indiferent de Controller

* @ControllerAdvice e si el un Controller si deci are nevoie de o clasa proprie.
* Daca folosim @ControllerAdvice, nu mai e nevoie de @ExceptionHandler
* In @ContollerAdvice cream metodele exact ca si la @ExceptionHandler. Nu uitam sa punem @ExceptionHandler inaintea fiecareia!



Creare:

1. Stergem toate @ExceptionHandler\

@ControllerAdvice  
public class ControllerException {  
 @ExceptionHandler  
 public ResponseEntity<StudentError> exception1(StudentException exception){  
 StudentError studentError = new StudentError();  
 studentError.setFirstName(exception.getMessage());  
 studentError.setLastName(String.*valueOf*(HttpStatus.*NOT\_FOUND*.value()));  
 studentError.setEmail(String.*valueOf*(System.*currentTimeMillis*()));  
  
 return new ResponseEntity<StudentError>(studentError,HttpStatus.*NOT\_FOUND*);  
 }  
 @ExceptionHandler  
 public ResponseEntity<StudentError> general(Exception exception){  
 StudentError studentError = new StudentError();  
 studentError.setFirstName(exception.getMessage());  
 studentError.setLastName(String.*valueOf*(HttpStatus.*BAD\_REQUEST*.value()));  
 studentError.setEmail(String.*valueOf*(System.*currentTimeMillis*()));  
  
 return new ResponseEntity<StudentError>(studentError,HttpStatus.*BAD\_REQUEST*);  
 }  
}

**Observatii!!!!**

* Intr-un block tray catch, ordinea in care punem blocurile try conteaza, asa cum trebuie sa fie de la cele mai putin generale la cel mai general. Insa, cand folosim @ExceptionHandler in interiorul la un Controller sau la @ControllerAdvice, ordinea lor nu conteaza, asa cum se va cauta intai o metoda ce are un parametru exact de tipul exceptiei aruncate, si apoi va cauta altul.
* Daca vom returna un String intr-o metoda din @ControllerAdvice, se va returna pagina jsp respectiva.
* Cat eram in RestController, puteam pune ca o metoda cu @ExceptinHandler sa returneze chiar si un obiect de al nostru, nu neaparat de tip ResponseEntity,caci oricum era transforma in JSON, insa in @ControllerAdvice asta deja nu mai merge.
* De ex in @RestController putem pune asa exception handler:
* @org.springframework.web.bind.annotation.ExceptionHandler  
  public StudentErrorResponse error(Exception e){  
   StudentErrorResponse error = new StudentErrorResponse();  
   error.setMessage("Unknown error");  
   error.setStatus(HttpStatus.*BAD\_REQUEST*.value());  
   error.setTimeStamp(System.*currentTimeMillis*());  
    
   return error;  
  }

Cand inca suntem in RestController, avem suport de la Jackson, si el va converti orice obiect in forma JSON

In @ControllerAdvice, chestiile nu mai stau asa. El deja e pentru orice Controller, deci nu numai pentru REST controller. Daca metoda de mai sus va fi pusa in @ControllerAdvice, nu se va returna o pagina cu date in JSON, ci o eroare de la tomcat. Insa, anume cu asta si ne ajuta obiectul ResponseEntity. El e cel care returneaza orice obiect in format JSON.

* Exceptia pentru paginile jsp negasite e

NoHandlerFoundException

Sau adaugam asta in application.properties

server.error.path=/error  
server.error.whitelabel.enabled=false

si acum /error request va fi automat chemat pentru orice invalid request

* E bine ca la @ExceptionHandler sa punem ca argument si exceptia ce o poate capta:
* @ExceptionHandler(ArrayIndexOutOfBoundsException.class)