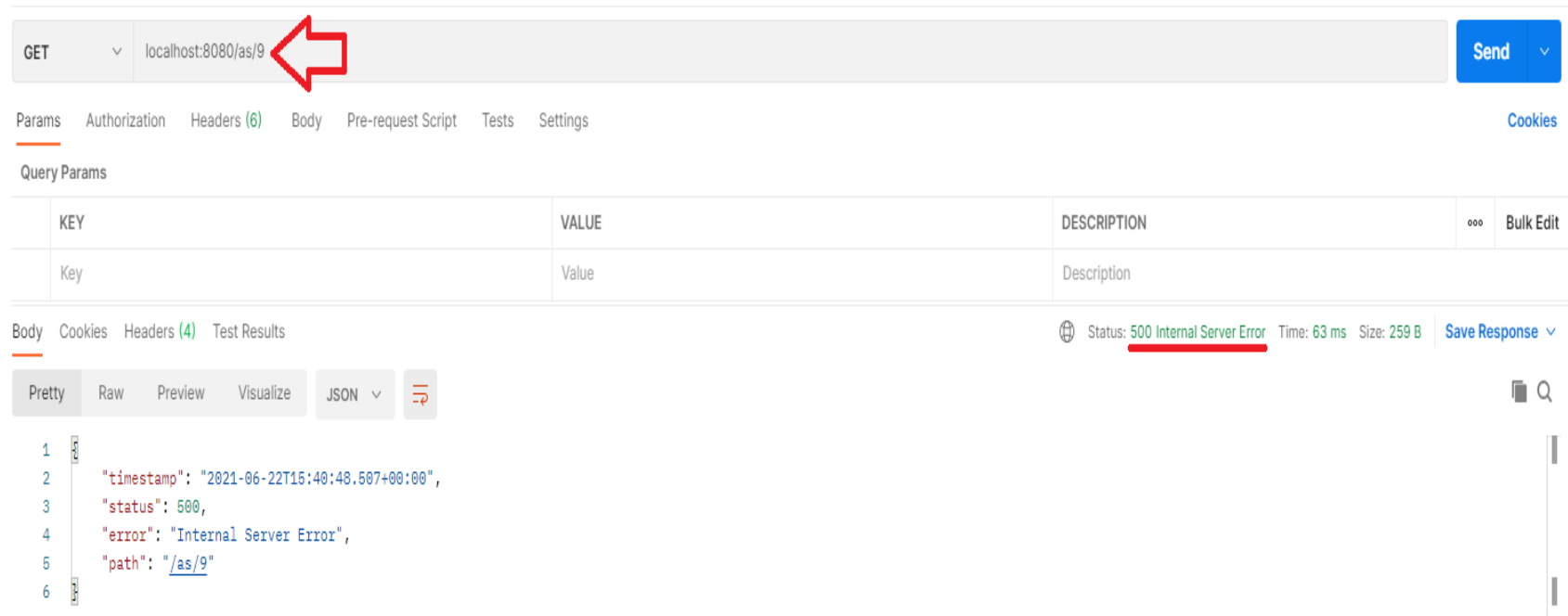


# Tratamento de erros (exceções)



# Introdução



The screenshot shows a REST client interface with a GET request to `localhost:8080/as/9`. A red arrow points to the URL. The response is a 500 Internal Server Error with a JSON body:

```
1 {
2   "timestamp": "2021-06-22T15:40:48.507+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "path": "/as/9"
6 }
```

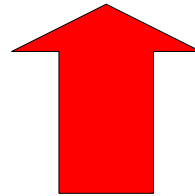
- Serviços implementados corretamente irão emitir códigos de erro condizentes com as situações de erro encontradas. O uso de exceções é fundamental para tratar esse tipo de questão.
- Na funcionalidade de obterPorId, por exemplo, se solicitarmos a busca de um objeto da classe A por um id inexistente (9 no exemplo acima) será emitido o código de erro “500” e essa não é uma estratégia interessante.



# Criando ObjetoNaoEncontradoException

```
// RuntimeException é uma exceção que o compilador não nos obriga tratar
public class ObjetoNaoEncontradoException extends RuntimeException{
    private static final long serialVersionUID = 1L;

    public ObjetoNaoEncontradoException(Object id) {
        super("Objeto não encontrado. ID = " + id);
    }
}
```



Camada de serviços



# Criando CorpoErroPadrao

```
public class CorpoErroPadrao implements Serializable{
    private static final long serialVersionUID = 1L;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss'Z'", timezone = "GMT")
    private Instant timestamp;
    private Integer status;
    private String error;
    private String message;
    private String path;

    public CorpoErroPadrao() {}

    public CorpoErroPadrao(Instant timestamp, Integer status, String error, String message, String path) {
        super();
        this.timestamp = timestamp;
        this.status = status;
        this.error = error;
        this.message = message;
        this.path = path;
    }

    public Instant getTimestamp() {}
    public void setTimestamp(Instant timestamp) {}
    public Integer getStatus() {}
    public void setStatus(Integer status) {}
    public String getError() {}
    public void setError(String error) {}
    public String getMessage() {}
    public void setMessage(String message) {}
    public String getPath() {}
    public void setPath(String path) {}
}
```

## Camada de Recursos

```
1 {}
2 "timestamp": "2021-06-22T15:40:48.507+00:00",
3 "status": 500,
4 "error": "Internal Server Error",
5 "path": "/as/9"
6 {}
```



# Tratador de Exceções

- **@ControllerAdvice** avisa ao framework que essa classe irá interceptar exceções geradas.
- **@ExceptionHandler** informa que exceção o método na sequência (objetoNaoEncontrado) irá tratar.

HttpStatus.NOT\_FOUND => Erro 404

```
@ControllerAdvice
public class TratadorExcecoesRecursos {

    @ExceptionHandler(ObjectoNaoEncontradoException.class)
    public ResponseEntity<CorpoErroPadrao> objetoNaoEncontrado(ObjectoNaoEncontradoException erro, HttpServletRequest request){
        String error = "Objeto não encontrado!!! ";
        HttpStatus status = HttpStatus.NOT_FOUND;
        CorpoErroPadrao corpoErroPadrao = new CorpoErroPadrao(Instant.now(), status.value(), error, erro.getMessage(), request.getRequestURI());
        return ResponseEntity.status(status).body(corpoErroPadrao);
    }
}
```



```
// RuntimeException é uma exceção que o compilador não nos obriga tratar
public class ObjetoNaoEncontradoException extends RuntimeException{
    private static final long serialVersionUID = 1L;

    public ObjetoNaoEncontradoException(Object id) {
        super("Objeto não encontrado. ID = " + id);
    }
}
```

# Disparando a exceção

```
@Service
public class ServicoA {

    // Fazendo a Injeção de Dependência
    @Autowired
    private RepositorioA repositorio;

    public List<A> obterTodos(){

    }

    public A obterPorId(Long id) {
        Optional<A> obj = repositorio.findById(id);
        return obj.orElseThrow(() -> new ObjetoNaoEncontradoException(id));
    }

    public A inserir(A a) {

    }

    public void excluir(Long id) {

    }

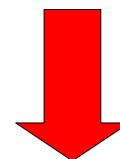
    public A update(Long id, A objeto_alterado) {

    }

    public void atualizarDados(A destino, A origem) {

    }
}
```

A exceção criada será tratada pelo  
**TratadorExcecoesRecursos**



- O método **orElseThrow** tenta obter o objeto da classe A pelo id informado, se não conseguir dispara a exceção **ObjetoNaoEncontradoException**



# Testando a aplicação

The screenshot shows a web browser interface with a GET request to `localhost:8080/as/9`. The response status is `404 Not Found`. The response body is displayed in JSON format:

```
{
  "timestamp": "2021-06-22T16:31:50Z",
  "status": 404,
  "error": "Objeto não encontrado!!! ",
  "message": "Objeto não encontrado. ID = 9",
  "path": "/as/9"
}
```

Red arrows point from the JSON fields to the corresponding code in the Java snippet below:

- `"timestamp": "2021-06-22T16:31:50Z",` points to `@ExceptionHandler`
- `"status": 404,` points to `HttpStatus.NOT_FOUND`
- `"error": "Objeto não encontrado!!! ",` points to `String error = "Objeto não encontrado!!! ";`
- `"message": "Objeto não encontrado. ID = 9",` points to `erro.getMessage()`
- `"path": "/as/9"` points to `request.getRequestURI()`

The Java code snippet is as follows:

```
// RuntimeException é uma exceção que o compilador não nos obriga tratar
public class ObjetoNaoEncontradoException extends RuntimeException{
    private static final long serialVersionUID = 1L;

    public ObjetoNaoEncontradoException(Object id) {
        super("Objeto não encontrado. ID = " + id);
    }
}

@ControllerAdvice
public class TratadorExcecoesRecursos {

    @ExceptionHandler(ObjetoNaoEncontradoException.class)
    public ResponseEntity<CorpoErroPadrao> objetoNaoEncontrado(ObjetoNaoEncontradoException erro, HttpServletRequest request){
        String error = "Objeto não encontrado!!! ";
        HttpStatus status = HttpStatus.NOT_FOUND;
        CorpoErroPadrao corpoErroPadrao = new CorpoErroPadrao(Instant.now(), status.value(), error, erro.getMessage(), request.getRequestURI());
        return ResponseEntity.status(status).body(corpoErroPadrao);
    }
}
```



# Tratando a exclusão de objetos A

- Se rodarmos a aplicação e tentarmos excluir um objeto inexistente (id = 5, por exemplo) recebemos o código de erro “500”.

The screenshot shows a REST client interface. At the top, a DELETE request is configured for the URL `localhost:8080/as/5`. A red arrow points to this URL. Below the request bar, the 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response indicates a 500 Internal Server Error. A second red arrow points to the status text 'Status: 500 Internal Server Error'.

DELETE `localhost:8080/as/5` Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

1  
2 `"timestamp": "2021-06-23T15:48:55.978+00:00",`  
3 `"status": 500,`  
4 `"error": "Internal Server Error",`  
5 `"path": "/as/5"`  
6

Status: 500 Internal Server Error Time: 452 ms Size: 259 B Save Response





# Tratando a exclusão de objetos A

```
@Service
public class ServicoA {

    // Fazendo a Injeção de Dependência
    @Autowired
    private RepositorioA repositorio;

    public List<A> obterTodos(){

    }

    public A obterPorId(Long id) {
        Optional<A> obj = repositorio.findById(id);
        return obj.orElseThrow(() -> new ObjetoNaoEncontradoException(id));
    }

    public A inserir(A a) {}

    public void excluir(Long id) {
        try {
            repositorio.deleteById(id);
        } catch (EmptyResultDataAccessException e) {
            throw new ObjetoNaoEncontradoException(id);
        }
    }

    public A update(Long id, A objeto_alterado) {}

    public void atualizarDados(A destino, A origem) {}
}
```

- Quando tentamos excluir um objeto de id inexistente geramos a exceção **EmptyResultDataAccessException**.
- Para facilitar o tratamento dessa exceção e por se tratar do mesmo tipo de erro (objeto inexistente) optamos por disparar uma exceção do tipo **ObjetoNaoEncontradoException**. Essa abordagem não é obrigatória, mas ao fazê-la repassamos o id para a exceção e uma mensagem mais ilustrativa poderá ser gerada.



# Tratando a exclusão de objetos A

The screenshot shows a REST client interface with a DELETE request to `localhost:8080/as/5`. The response status is `404 Not Found`. The response body is displayed in JSON format:

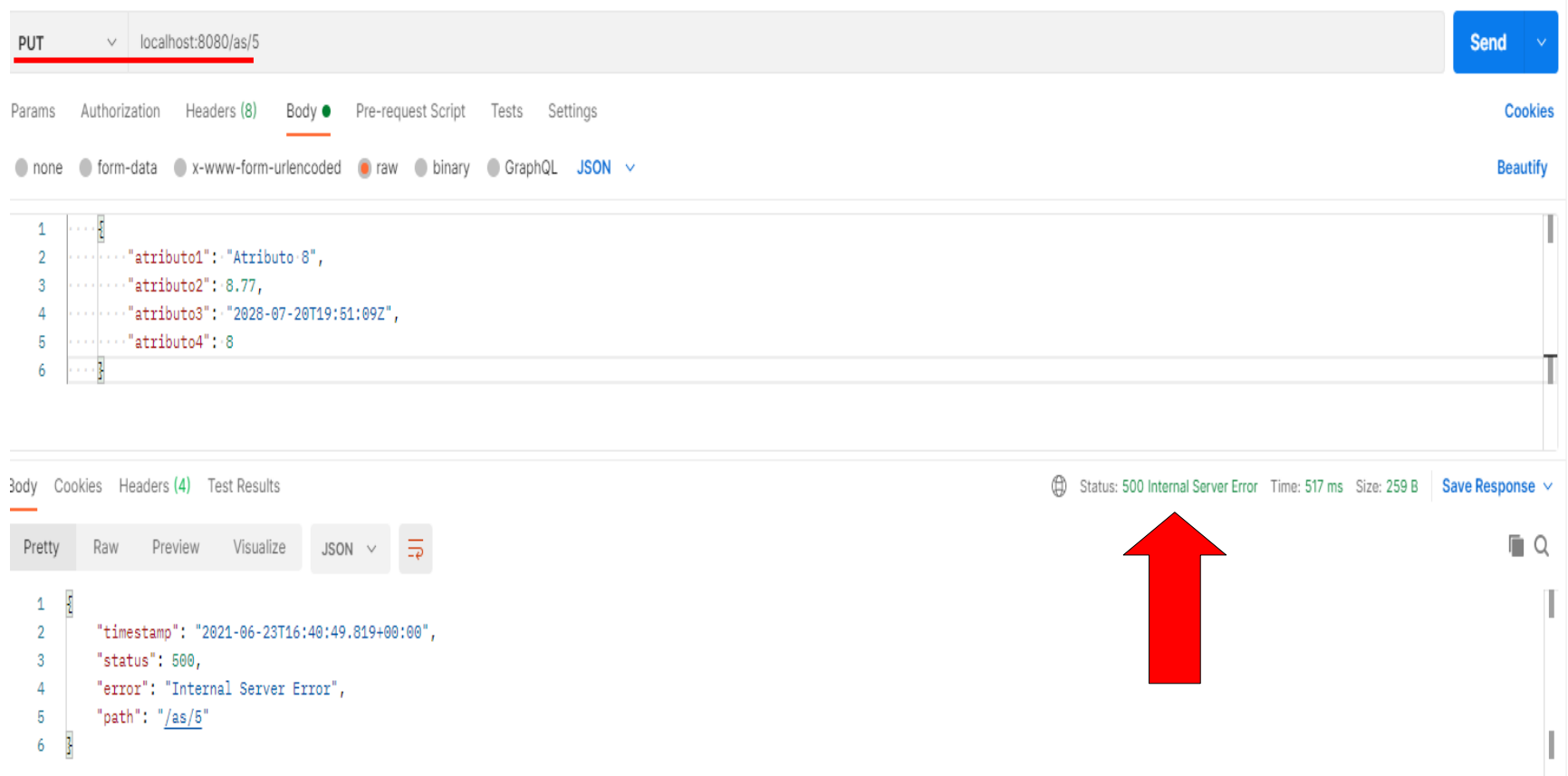
```
1 {
2   "timestamp": "2021-06-23T16:18:38Z",
3   "status": 404,
4   "error": "Objeto não encontrado!!! ",
5   "message": "Objeto não encontrado. ID = 5",
6   "path": "/as/5"
7 }
```

Two red arrows highlight the status and the error message in the response body.



# Tratando a atualização de objetos A

- Se rodarmos a aplicação e tentarmos atualizar um objeto inexistente (id = 5, por exemplo) recebemos o código de erro “500”.



# Tratando a atualização de objetos A

```
@Service
public class ServicoA {

    // Fazendo a Injeção de Dependência
    @Autowired
    private RepositorioA repositorio;

    public List<A> obterTodos(){

    }

    public A obterPorId(Long id) {

    }

    public A inserir(A a) {

    }

    public void excluir(Long id) {
        try {
            repositorio.deleteById(id);
        } catch (EmptyResultDataAccessException e) {
            throw new ObjetoNaoEncontradoException(id);
        }
    }

    public A update(Long id, A objeto_alterado) {
        try {
            A a = repositorio.getById(id);
            atualizarDados(a, objeto_alterado);
            return repositorio.save(a);
        } catch (EntityNotFoundException e) {
            throw new ObjetoNaoEncontradoException(id);
        }
    }

    public void atualizarDados(A destino, A origem) {

    }
}
```

- Da mesma forma que fizemos na exclusão, devemos capturar a exceção gerada pelo serviço. Nesse caso **EntityNotFoundException**.
- Como temos o mesmo tipo de erro (Objeto Não Encontrado) optamos por disparar **ObjetoNaoEncontradoException**.

Essa abordagem não é obrigatória, mas ao fazê-la repassamos o id para a exceção e uma mensagem mais ilustrativa poderá ser gerada.



# Tratando a atualização de objetos A

The screenshot shows a REST client interface. The top bar indicates a PUT request to `localhost:8080/as/5`, with a red arrow pointing to the URL. The 'Body' tab is selected, showing a JSON object with four attributes: `atributo1`, `atributo2`, `atributo3`, and `atributo4`. The bottom section shows the response, which is a 404 Not Found status. A red arrow points to the status text. The response body is a JSON object containing a timestamp, status, error message, and path.

```
PUT localhost:8080/as/5
```

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "atributo1": "Atributo 8",
3   "atributo2": 8.77,
4   "atributo3": "2028-07-20T19:51:09Z",
5   "atributo4": 8
6 }
```

Body Cookies Headers (5) Test Results

Status: 404 Not Found Time: 469 ms Size: 315 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2021-06-23T16:47:53Z",
3   "status": 404,
4   "error": "Objeto não encontrado!!! ",
5   "message": "Objeto não encontrado. ID = 5",
6   "path": "/as/5"
7 }
```



# Dúvidas?

