02

(DESAFIO) Tratamento especial para InvocationTargetException

E então, meu aluno? Conseguiu chegar na solução do desafio visto durante a aula? Não? Então não tem problema! Eu irei tirá-lo das trevas!

O que gostaríamos de fazer é ter um método chamado `comTratamentoDeExcecao()` que receberia como parâmetro uma expressão lambda. A qual, por sua vez, receberia o método em questão e uma exceção do tipo `InvocationTargetException`, declarando a seguir como deveria ser feito o tratamento do erro. Justamente como visto durante a aula!

```
public class Alurator {

    // código anterior omitido.

    public Object executa(String url) {
        Object retorno =
            new Reflexao()
                .refleteClasse(basePackage + nomeControle)
                .criaInstancia(container)
                .getMetodo(nomeMetodo, queryParams)
                .comTratamentoDeExcecao((metodo, e) -> {
                    System.out.println("Erro no método " + metodo.getName() + " da classe " + metodo
                    throw new RuntimeException("ERRO!");
                })
                .invocar();

        return retorno;
    }
}
```

Desse modo, precisamos responder duas perguntas! Onde esse método deve ser implementado e que tipo de parâmetro ele irá receber.

A primeira questão é bem simples de responder! Basta observar que o método `getMetodo()` retorna um objeto do tipo `ManipuladorMetodo`. Sendo assim, se queremos que tal objeto tenha um método chamado `comTratamentoDeExcecao()`, é justamente na classe `ManipuladorMetodo` que precisamos implementá-lo!

Agora, a segunda questão para ser respondida precisa de uma perspicácia maior quanto às interfaces funcionais que foram introduzidas no JDK 8! Perceba que a nossa expressão lambda recebe dois parâmetros de tipos distintos! O método que está sendo invocado, portanto um objeto da classe `Method` e a exceção que deverá ser tratada que nesse caso é um objeto do tipo `InvocationTargetException`. Por fim, essa expressão lambda deve ser capaz de dar um retorno de um terceiro tipo, o qual, nesse caso, representamos com o lançamento de uma nova exceção do tipo `RuntimeException`.

De posse dessas características, precisamos apenas saber se há uma interface funcional que as atenda! É aí que entra a perspicácia! O aluno que estiver mais por dentro desse assunto, saberá que, para a nossa alegria, há uma interface que atende todas as nossas necessidades! E ela se chama `BiFunction<T, U, R>` (link do Javadoc (https://docs.oracle.com/javase/10/docs/api/java/util/function/BiFunction.html))! Essa interface funcional possui um método `apply(T t, U u)` que recebe dois objetos de tipos genéricos `T` e `U` e retorna um terceiro tipo `R`. Em outras palavras, é esse método que deve ser invocado para que nossa expressão lambda funcione!

Sendo assim, a implementação na classe `ManipuladorMetodo` deve ser como o a seguir!

```
public class ManipuladorMetodo {
    private BiFunction<Method, InvocationTargetException, Object> tratamentoExcecao;

    // código anterior omitido.

    public ManipuladorMetodo comTratamentoDeExcecao(BiFunction<Method, InvocationTargetException, Ob
        this.tratamentoExcecao = tratamentoExcecao;
        return this;
    }

    // código posterior omitido.

    public Object invocar() {
        List<Object> parametros = new ArrayList<>();
        Stream.of(metodo.getParameters())
                .forEach(p -> parametros.add(args.get(p.getName())));
        try {
            return metodo.invoke(alvo, parametros.toArray());
        } catch (IllegalAccessException
                | IllegalArgumentException e) {

            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (InvocationTargetException e) {

            // tratamento especial e customizado da exceção.
            if (tratamentoExcecao != null) {
                return tratamentoExcecao.apply(metodo, e);
            }

            e.printStackTrace();
            throw new RuntimeException("Erro no método!", e.getTargetException());
        }
    }
}
```

E então, meu aluno? O que achou? Bem bacana, não? Dessa maneira, conseguimos resolver de forma elegante um problema importante que é o tratamento especial para a `InvocationTargetException` e ainda, **como extra, afinal de contas você está tendo aula com o instrutor Gabriel Leite**, aprendemos um pouco mais sobre as interfaces funcionais introduzidas no JDK 8!

Foi essa a solução que encontrou? Se conseguiu resolver o problema de forma diferente, conte para mim e outros colegas da comunidade como fez mandando sua resposta para o fórum!