Nolix

Nolix error handling

2018-09-20

Table of contents

1 Nolix validator

1.1 What, why and how

What the Nolix validator is

The Nolix validator is a Java artifact, that provides methods to validate **incoming** arguments.

Why to use the Nolix validator

- The Nolix validator produces **uniform** and **constistent** error messages.
- The Nolix validator can validate arguments on **various** properties.
- The Nolix validator calls can be written in **very legible** code.

How to use the Nolix validator

To use the Nolix validator, import the Nolix library into your project. Then you can access the Nolix validator in your project. The Nolix validator can be found in the ‘ch.nolix.primitive.validator2’ package.

1.2 How to validate arguments on required properties

To validate an argument, call the static method ‘suppose’ of the validator giving your argument as parameter. Then call one of the available validation methods. If the argument does not fulfil the required properties, the validator will throw a suitable exception with a suitable error message.

Example

Lets create a class ‘Bag’ that just defines with a bag with one object as content. The content is given by the constructor and must be an instance (not null). In the constructor, let the validator validate whether the given content is an instance (not null).

```
import ch.nolix.primitive.validator2

public class Bag {

  private final Object content;

  //constructor
  public Bag(Object content) {

    Validator.suppose(content).isInstance();

    this.content = content;
  }

  public Object getContent() {
    return content;
  }
}
```

If the given content is null, the validator will throw a NullArgumentException.

The error message of the NullArgumentException will be:

“The given argument is null.”

1.3 How to validate strings

The validator can validates strings specifically. Strings can be validated whether they are empty, start with a letter, have a maximum length, etc.

Example

```
public void setName(String name) {

  Validator.suppose(name).isNotEmpty();

  …
}
```

If the given name is an empty string, the validator will throw an EmptyArgumentException.

The error message of the EmptyArgumentException will be:

“The given String ‘’ is empty.”

If the given name is null, the validator will throw a NullArgumentException.

1.4 How to validate numbers

The validator can validate numbers specifically. Numbers can be validated whether they are negative, positive, smaller than a given value, bigger than a given value, etc.

Example

```
public void setAmount(int amount) {

  Validator.suppose(amount).isPositive();

  …
}
```

If the given amount is not positive, the validator will throw a NonPositiveArgumentException.

For the amount -25, the error message of the NonPositiveArgumentException will be:

"The given Integer '-25' is not positive."

Example

```
public void buyPencils(int amount) {

  Validator.suppose(amount).isBetween(100, 10000);

  …
}
```

If the given amount is not between 100 and 10000, the validator will throw an OutOfRangeArgumentException.

For the value 50, the error message of the OutOfRangeException will be:

"The given Integer '50' is not between 100 and 10000."

1.5 How to validate collections

The validator provides 2 kinds how to validate collections.

- Validate a **collection itself**.
- Validate the **elements** of a collection.

Example (Validate an integer collection itself)

```
Public void saveHitPoints(int[] hitPoints) {

  Validator.suppose(hitPoints).isNotEmpty();

  …

}
```

If the given hit points array is empty, the validator will throw an EmptyArgumentException.

The error message of the EmptyArgumentException will be:

"The given array is empty."

If the given hit points array is null, the validator will throw a NullArgumentException.

Example (Validates the elements of an integer collection)

```
public void saveHitPoints(int[] hitPoints) {

  Validator.supposeTheInts(hitPoints).areNotNegative();

  …
}
```

If one of the given hit points is negative, the validator will throw a NegativeArgumentException.

For the value -10 as the 5th argument, the error message of the NegativeArugmentException will be:

"The given 5th argument '-10' is negative."

If the given hit points array is null, the validator will throw a NullArgumentException.

2 Nolix exceptions

2.1 What, why and how

What Nolix exceptions are

Nolix exceptions are different exceptions for different cases.

Why to use Nolix exceptions

If the code throws exceptions directly, then the Nolix exceptions offers the following advantages.

- Nolix exceptions produce **uniform** and **consistent** error messages.
- Nolix exceptions provide **suitable constructors**, e.g. for taking information to create detailed error messages.
- There are **suitable types** Nolix exceptions for the most situations.

How to use Nolix excpetions

To use the Nolix exceptions, import the Nolix library into your project. Then you can access the Nolix exceptions in your project. The Nolix exceptions can be found in the following packages.

- ch.nolix.primitive.invalidArgumentExceptions
- ch.nolix.primitive.invalidStateExceptions

2.2 Categories of exceptions

There are 2 categories of exceptions.

- InvalidArgumentExceptions are intended to be thrown when a given argument does not fulfil the required properties.
- InvalidStateArgumentException are intended to be thrown when a current instance is in a state, where a specific member method is not allowed to be called on.

2.3 InvalidArgumentExceptions

For any arguments

Exception	Purpose
EmptyArgumentException	Intended to be thrown when an argument is empty, e.g. strings or containers.
NullArgumentException	Intended to be thrown when an argument is null.

For numbers

Exception	Purpose
NegativeArgumentException	Intended to be thrown when number is negative.
OutOfRangeArgumentException	Intended to be thrown when a number is not in a given range.
SmallerArgumentException	Intended to be thrown when a number is smaller than a given limit.
ZeroArgumentException	Intended to be thrown when a number is 0.

2.4 InvalidStateExceptions

Exception	Purpose
ClosedStateException	Intendend to be thrown when an instance is closed and a specific member method is called on it at the wrong time.
EmptyStateException	Intended to be thrown when an instance is empty and a specific method is called on it at the wrong time.