How to check if an enum value exists in Java

Updated on November 09, 2022

▶ **IN THIS ARTICLE** 👇

There are multiple ways to check if an enum contains the given string value in Java. You can use the `valueOf()` to convert the string into an enum value in Java. If the string is a valid enum value, the `valueOf()` method returns an enum object. Otherwise, it throws an exception.

Let us say we have the following enum representing the days of the week:

```
public enum Weekday {
    MONDAY(1, "Monday"),
    TUESDAY(2, "Tuesday"),
```

```java
    WEDNESDAY(3, "Wednesday"),
    THURSDAY(4, "Thursday"),
    FRIDAY(5, "Friday"),
    SATURDAY(6, "Saturday"),
    SUNDAY(7, "Sunday");

    private final int number;
    private final String value;

    Weekday(int number, String value) {
        this.number = number;
        this.value = value;
    }

    public int getNumber() {
        return number;
    }

    public String getValue() {
        return value;
    }
}
```

Search an enum value by name

Since the `` `enum` `` type is a special data type in Java, the name of each enum is constant. For example, the name of `` `Weekday.FRIDAY` `` is `` `FRIDAY` ``.

To search the enum by its name, you can use the `` `valueOf()` `` method, as shown below:

```java
String str = "FRIDAY";

Weekday day = Weekday.valueOf(str.toUpperCase());
System.out.println(day); // FRIDAY
```

The `` `valueOf()` `` method works as long as you pass a string that matches an enum name. If the given string does not match an enum name, an `` `IllegalArgumentException` `` is thrown:

```java
// Fail due to case mismatch
Weekday monday = Weekday.valueOf("Monday");
```

```java
// Fail due to invalid value
Weekday invalidStr = Weekday.valueOf("Weekend");
```

To handle exceptions gracefully, let us add a static function called `findByName()` to `Weekday` that searches the enum by name and returns `null` if not found:

```java
public enum Weekday {
    // ...

    public static Weekday findByName(String name) {
        for (Weekday day : values()) {
            if (day.name().equalsIgnoreCase(name)) {
                return day;
            }
        }
        return null;
    }
}
```

The `findByName()` method uses the `values()` method to iterate over all enum values and returns an enum object if the given string matches an enum name. Otherwise, it returns `null` as shown below:

```
System.out.println(Weekday.findByName("Monday")); // MONDAY
System.out.println(Weekday.findByName("friday")); // FRIDAY
System.out.println(Weekday.findByName("Weekday")); // null
```

Search an enum by value

To search an enum by value, you can use the same `valueOf()` method you used above. This time, instead of using the `name()` method, use the `getValue()` method, as shown below:

```
public enum Weekday {
    // ...

    public static Weekday findByValue(String value) {
        for (Weekday day : values()) {
            if (day.getValue().equalsIgnoreCase(value)) {
                return day;
            }
        }
        return null;
    }
}
```

The `findByValue()` method works to `findByName()` and returns an enum object that matches the enum value. So for `Tuesday`, you will get `Weekday.TUESDAY`. If the given value is invalid, the `findByValue()` method returns `null` as shown below:

```
System.out.println(Weekday.findByValue("Tuesday")); // TUESDAY
System.out.println(Weekday.findByValue("friday")); // FRIDAY
System.out.println(Weekday.findByValue("Fri")); // null
```

Search an enum by an integer value

You can also write a function to search an enum by an integer value. For example, to search a `` `Weekday` `` by the `` `number` `` field, you can use the following code:

```
public enum Weekday {
    // ...

    public static Weekday findByNumber(int number) {
        for (Weekday day : values()) {
            if (day.getNumber() == number) {
                return day;
            }
        }
        return null;
    }
}
```

The `` `findByNumber()` `` method compares the given number with the day's number. If the number matches, it returns the matched object. Otherwise, it returns `` `null` `` as shown below:

```java
System.out.println(Weekday.findByNumber(3)); // WEDNESDAY
System.out.println(Weekday.findByNumber(5)); // FRIDAY
System.out.println(Weekday.findByNumber(9)); // null
```

Search an enum using Java 8 Streams

You can also use Java 8 Streams to simplify the enum search. Let us rewrite the above `` `findByNumber()` `` method using streams:

```java
public static Optional<Weekday> findByNumber(int number) {
    return Arrays.stream(values()).filter(weekday -> weekday.getNumber() == number)
            .findFirst();
}
```

The above example code looks different from the previous codes because we have used Java 8 streams to implement the search.

Also, instead of returning the enum itself, an `Optional` value of the `Weekday` is returned. For the `null` value, the `findByNumber()` method returns an empty `Optional`.

Here is an example that uses the above method to check if the enum exists:

```
Optional<Weekday> weekday = Weekday.findByNumber(6);
if (weekday.isPresent()) {
    System.out.println("The number matches " + weekday.get().name());
} else {
    System.out.println("Number does not match.");
}

// The number matches SATURDAY
```

Throwing an exception if the enum is not found

You can also throw an exception instead of returning `null` or empty `Optional`. Let us again rewrite the `findByNumber()` to throw an `IllegalArgumentException` if the enum is not found:

```java
public static Weekday findByNumber(int number) {
    return Arrays.stream(values()).filter(weekday -> weekday.getNumber() == number)
            .findFirst().orElseThrow(IllegalArgumentException::new);
}
```

The following example uses the above `findByNumber()` method to search an enum by an integer value:

```java
System.out.println(Weekday.findByNumber(5)); // FRIDAY
System.out.println(Weekday.findByNumber(8));
// Exception in thread "main" java.lang.IllegalArgumentException
```

#Java

You might also like...

[Calculate days between two OffsetDateTime objects in Java]

[Calculate days between two ZonedDateTime objects in Java]

[Calculate days between two LocalDateTime objects in Java]

[Calculate days between two LocalDate objects in Java]

[How to iterate over enum values in Java]

Buy me a coffee ☕

If you enjoy reading my articles and want to help me out paying bills, please consider buying me a coffee ($5) or two ($10). I will be highly grateful to you ✌️

Enter the number of coffees below:

2 Buy

✨ Learn to build modern web applications using JavaScript and Spring Boot

I started this blog as a place to share everything I have learned in the last decade. I write about modern JavaScript, Node.js, Spring Boot, core Java, RESTful APIs, and all things web development.

The newsletter is sent every week and includes **early access** to clear, concise, and easy-to-follow tutorials, and other stuff I think you'd enjoy! No spam ever, unsubscribe at any time.

Your Email Address Subscribe

Atta.

Maker. Developer. Writer.

Products	Links	Follow Me	Others
DomBuck	Articles	Twitter	Privacy Policy
StartupBase	Topics	LinkedIn	Cookies
AcquireBase	Advertise	GitHub	Disclaimer
Protips	Contact	RSS Feed	