

# ValidationUtil - Missing Hierarchy

---

[Link to chat 1](#)

[Link to chat 2](#)

[Manual Refactoring](#)

file: books-web-common/src/main/java/com/sismics/rest/util/ValidationUtil.java

## Chat 1: Before being told that it was a Missing Hierarchy design smell:

The `ValidationUtil` class indeed exhibits some design smells or areas for improvement. Here are a few identified issues along with potential refactorings:

### 1. Single Responsibility Principle (SRP) Violation:

- The class is doing too much: it validates string lengths, checks for email patterns, validates URLs, parses dates, and validates locale and theme IDs. It's better to split these responsibilities into separate classes.

### 2. Dependency Injection:

- The class directly instantiates `LocaleDao` and `ThemeDao`. Instead, it should rely on these dependencies being injected, making it more flexible and easier to test.

### 3. Error Handling:

- The use of checked exceptions (`JSONException`) for validation purposes is unusual and complicates the code. Using unchecked exceptions (like `IllegalArgumentException`) would be more appropriate for validation.

### 4. Hardcoded Messages:

- Error messages are hardcoded, making it harder to internationalize or customize them. Using externalized message resources or at least constants for error messages would be better.

### 5. Regex Patterns:

- Regex patterns like `EMAIL_PATTERN`, `HTTP_URL_PATTERN`, and `ALPHANUMERIC_PATTERN` could be moved to a configuration file or constants for better maintainability.

Here's a refactored version addressing these issues:

```
package com.sismics.rest.util;

import java.util.regex.Pattern;

public class ValidationUtil {
    private static final Pattern EMAIL_PATTERN =
Pattern.compile(".*@.*\\..*");
    private static final Pattern HTTP_URL_PATTERN =
```

```

Pattern.compile("https?://.+");
    private static final Pattern ALPHANUMERIC_PATTERN = Pattern.compile("[a-zA-Z0-9_]+");

    public static void validateRequired(Object s, String name) {
        if (s == null) {
            throw new IllegalArgumentException(name + " must be set");
        }
    }

    public static String validateLength(String s, String name, int
lengthMin, int lengthMax, boolean nullable) {
        s = s != null ? s.trim() : null;
        if (nullable && (s == null || s.isEmpty())) {
            return s;
        }
        validateRequired(s, name);
        if (s.length() < lengthMin) {
            throw new IllegalArgumentException(name + " must be more than "
+ lengthMin + " characters");
        }
        if (lengthMax > 0 && s.length() > lengthMax) {
            throw new IllegalArgumentException(name + " must be less than "
+ lengthMax + " characters");
        }
        return s;
    }

    public static String validateLength(String s, String name, int
lengthMin, int lengthMax) {
        return validateLength(s, name, lengthMin, lengthMax, false);
    }

    public static String validateStringNotBlank(String s, String name) {
        return validateLength(s, name, 1, Integer.MAX_VALUE);
    }

    public static void validateHexColor(String s, String name, boolean
nullable) {
        validateLength(s, name, 7, 7, nullable);
    }

    public static void validateEmail(String s, String name) {
        if (!EMAIL_PATTERN.matcher(s).matches()) {
            throw new IllegalArgumentException(name + " must be an email");
        }
    }

    public static String validateHttpUrl(String s, String name) {
        s = s != null ? s.trim() : null;
        if (s == null || !HTTP_URL_PATTERN.matcher(s).matches()) {
            throw new IllegalArgumentException(name + " must be an HTTP(s)
URL");
        }
    }

```

```

        return s;
    }

    public static void validateAlphanumeric(String s, String name) {
        if (!ALPHANUMERIC_PATTERN.matcher(s).matches()) {
            throw new IllegalArgumentException(name + " must have only
alphanumeric or underscore characters");
        }
    }
}

```

This refactoring separates concerns, simplifies error handling, and makes the code more maintainable and testable. Additionally, it removes unnecessary dependencies and improves error message handling.

---

## Chat 2: After being told that it was a Missing Hierarchy design smell:

To address the missing hierarchy design smell, we can refactor the `ValidationUtil` class to better organize the validation methods into separate classes based on their responsibilities. We can create validator classes for different types of validations, such as string validations, URL validations, email validations, etc. Each validator class can implement a common interface or extend a common abstract class to ensure consistency and ease of use.

Here's how we can refactor the `ValidationUtil` class:

1. Create an interface called `Validator` that defines a method for validation.
2. Create validator classes for string validation, URL validation, email validation, etc., implementing the `Validator` interface.
3. Modify the `ValidationUtil` class to use instances of these validator classes for different types of validations.

Here's a possible refactored version:

```

package com.sismics.rest.util;

import org.codehaus.jettison.json.JSONException;

/**
 * Interface for validators.
 */
public interface Validator<T> {
    void validate(T value, String name) throws JSONException;
}

```

```

package com.sismics.rest.util;

import com.sismics.rest.exception.ClientException;
import org.apache.commons.lang.StringUtils;

```

```

import org.codehaus.jettison.json.JSONException;

import java.text.MessageFormat;
import java.util.regex.Pattern;

/**
 * Validator for string parameters.
 */
public class StringValidator implements Validator<String> {
    private static final Pattern ALPHANUMERIC_PATTERN = Pattern.compile("[a-zA-Z0-9_]+");

    @Override
    public void validate(String value, String name) throws JSONException {
        value = StringUtils.strip(value);
        if (StringUtils.isEmpty(value)) {
            throw new ClientException("ValidationError",
MessageFormat.format("{0} must be set", name));
        }
    }

    public void validateNotBlank(String value, String name) throws
JSONException {
        validate(value, name);
        if (value.trim().isEmpty()) {
            throw new ClientException("ValidationError",
MessageFormat.format("{0} must not be blank", name));
        }
    }

    public void validateAlphanumeric(String value, String name) throws
JSONException {
        validateNotBlank(value, name);
        if (!ALPHANUMERIC_PATTERN.matcher(value).matches()) {
            throw new ClientException("ValidationError",
MessageFormat.format("{0} must have only alphanumeric or underscore
characters", name));
        }
    }
}

```

```

package com.sismics.rest.util;

import com.sismics.rest.exception.ClientException;
import org.codehaus.jettison.json.JSONException;

import java.text.MessageFormat;
import java.util.regex.Pattern;

/**
 * Validator for email addresses.

```

```

*/
public class EmailValidator implements Validator<String> {
    private static final Pattern EMAIL_PATTERN =
Pattern.compile(".*@.*\\.\\..*");

    @Override
    public void validate(String value, String name) throws JSONException {
        if (!EMAIL_PATTERN.matcher(value).matches()) {
            throw new ClientException("ValidationError",
MessageFormat.format("{0} must be an email", name));
        }
    }
}

```

```

package com.sismics.rest.util;

import com.sismics.rest.exception.ClientException;
import org.codehaus.jettison.json.JSONException;

import java.text.MessageFormat;
import java.util.regex.Pattern;

/**
 * Validator for HTTP URLs.
 */
public class HttpUrlValidator implements Validator<String> {
    private static final Pattern HTTP_URL_PATTERN =
Pattern.compile("https?://.*");

    @Override
    public void validate(String value, String name) throws JSONException {
        value = StringUtils.strip(value);
        if (!HTTP_URL_PATTERN.matcher(value).matches()) {
            throw new ClientException("ValidationError",
MessageFormat.format("{0} must be an HTTP(s) URL", name));
        }
    }
}

```

With this refactoring, each validation type has its own validator class, making the code more modular, maintainable, and easier to extend. You can further expand this structure by adding validators for other types of validations as needed.

---

Here, we see that the LLM was not able to figure out that it was a Missing Hierarchy design smell, and thus was only able to make minor changes for the refactoring - in a completely irrelevant manner.

However, when the issues and the location were highlighted, the LLM performed much better and was able to show successful rectification of the issue by forming a decent hierarchy with no other information.