Java Class Responsibility

I have a Java class in my project that I want to follow Java Solid Principles. Infer the class's responsibility and look for ways to improve it, ONLY if applicable.

```
package com.toolstore.demo.exception;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class ErrorResponse {
    private String message;

    public ErrorResponse(String message) {
        this.message = message;
    }

}
```

RouteLLM

Based on the Toolstore specification.txt and the Java code provided, the ErrorResponse class is intended to serve as a Data Transfer Object (DTO) for returning error details to the client when validation requirements (like invalid rental days or discount percentages) are not met.

While the class is simple, we can improve its adherence to SOLID principles—specifically the **Single Responsibility Principle (SRP)** and **Open/Closed Principle (OCP)**—to make it more robust for a production-grade API.

Inferred Responsibility

The current responsibility is to encapsulate a single error message string. However, in a RESTful system like a tool store rental application, an error response often needs to provide more context (such as timestamps or specific error codes) to help the client handle the exception properly.

Recommended Improvements

1. **Single Responsibility Principle (SRP):** Currently, the class only holds a message. To better serve its responsibility of "communicating error state," it should include a timestamp and potentially a status code. This ensures the client has a standardized way to log or display the error.
2. **Immutability:** DTOs used for responses should generally be immutable. Removing @Setter and using @Value or final fields prevents the response object from being modified after it is created, which is a best practice for thread safety and data integrity.
3. **Open/Closed Principle (OCP):** By making the class more descriptive, you allow the system to be "open for extension" (e.g., adding a list of field-specific validation errors) without changing the fundamental way the application handles the base ErrorResponse.

Improved Implementation

```
package com.toolstore.demo.exception;

import lombok.Getter;
import java.time.LocalDateTime;
import java.util.Collections;
import java.util.List;

@Getter
public class ErrorResponse {
   private final String message;
   private final LocalDateTime timestamp;
   private final List<String> details;

   // Primary constructor for simple messages
   public ErrorResponse(String message) {
      this(message, Collections.emptyList());
   }

   // Extended constructor for validation details (e.g., multiple invalid fields)
   public ErrorResponse(String message, List<String> details) {
      this.message = message;
      this.details = details;
      this.timestamp = LocalDateTime.now();
   }
}
```

Why this is better for your Toolstore project:

- **Validation Context:** If a user enters a discount of 101% and a rental day count of 0, you can now return both errors in the details list rather than just the first one encountered.
- **Standardization:** Including a timestamp helps in debugging logs when customers report issues with the checkout process.
- **Safety:** Using final fields and removing setters ensures that once an error is generated during the checkout validation, it cannot be accidentally altered by other parts of the code.