Exception Handling Review

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 org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import
org.springframework.web.bind.MethodArgumentNotValidException
;
import
org.springframework.web.bind.annotation.ExceptionHandler;
import
org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.stream.Collectors;

@RestControllerAdvice
public class GlobalExceptionHandler {

   @ExceptionHandler(IllegalArgumentException.class)
   public ResponseEntity
handleValidationException(IllegalArgumentException ex) {
       System.err.println("Checkout Validation Failed: " +
ex.getMessage());
       return ResponseEntity.badRequest().body(new
ErrorResponse(ex.getMessage()));
   }

   @ExceptionHandler(MethodArgumentNotValidException.class)
   public ResponseEntity
handleMethodArgumentNotValid(MethodArgumentNotValidExceptio
n ex) {
       String message = ex.getBindingResult()
               .getFieldErrors()
               .stream()
               .map(FieldError::getDefaultMessage)
               .collect(Collectors.joining(", "));

       System.err.println("Request Body Invalid: " + message);
       return ResponseEntity.badRequest().body(new
```

ErrorResponse(message));
}

@ExceptionHandler(Exception.class)
public ResponseEntity handleGeneralException(Exception ex) {
System.err.println("An unexpected error occurred: " +
ex.getMessage());
ex.printStackTrace(); // Useful for a console-based demo
return ResponseEntity.internalServerError().body(new
ErrorResponse("An internal error occurred."));
}
}```

RouteLLM

Based on the provided code and the **Toolstore specification**, the GlobalExceptionHandler class is responsible for intercepting and formatting error responses across the application. This ensures that validation failures (like the required "Rental day count" or "Discount percent" checks) result in user-friendly messages rather than raw stack traces.

In terms of **SOLID Principles**, specifically the **Single Responsibility Principle (SRP)** and **Dependency Inversion Principle (DIP)**, there are a few areas where the class can be improved to be more robust and maintainable.

Analysis of SOLID Improvements

1. Single Responsibility Principle (SRP)

The current class handles three distinct things: intercepting exceptions, logging/printing to the console, and constructing the ErrorResponse body. While a @RestControllerAdvice is naturally a "coordinator," the use of System.err.println and ex.printStackTrace() couples the error handling logic with a specific console-based logging implementation.

2. Dependency Inversion Principle (DIP)

The class currently depends on the concrete ErrorResponse class directly. While common in small demos, in a larger system, you might want to decouple the error structure from the handler to allow for different response formats without changing the handler logic.

3. Open/Closed Principle (OCP)

If the Toolstore adds specific domain exceptions (e.g., InvalidRentalDurationException or ToolNotFoundException), you would have to keep modifying this class to add new @ExceptionHandler methods.

Recommended Improvements

Refactor Logging and Response Logic

Instead of using System.err, use a proper SLF4J Logger. Additionally, to better follow SRP, you can ensure the ErrorResponse is a simple DTO and the handler focuses strictly on the mapping logic.

Specific Exception Handling

The specification mentions specific validation requirements (Rental days < 1, Discount > 100). While IllegalArgumentException works, creating a custom ValidationException allows you to separate "Business Rule Violations" from general "Illegal Arguments" that might occur deeper in the framework.

Improved Implementation

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

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.time.LocalDateTime;
import java.util.stream.Collectors;

@RestControllerAdvice
public class GlobalExceptionHandler {

    private static final Logger logger =
LoggerFactory.getLogger(GlobalExceptionHandler.class);

    // Handles Business Rule Validations (e.g., Rental days < 1, Discount % range)
    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<ErrorResponse>
handleBusinessValidation(IllegalArgumentException ex) {
        logger.error("Checkout Validation Failed: {}", ex.getMessage());
        return buildResponse(ex.getMessage(), HttpStatus.BAD_REQUEST);
    }

    // Handles JSR-303 / Spring Boot Request Body Validations
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse>
handleMethodArgumentNotValid(MethodArgumentNotValidException ex) {
        String message = ex.getBindingResult()
                .getFieldErrors()
                .stream()
                .map(FieldError::getDefaultMessage)
                .collect(Collectors.joining(", "));

        logger.error("Request Body Invalid: {}", message);
        return buildResponse(message, HttpStatus.BAD_REQUEST);
    }

    // Catch-all for unexpected internal errors
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGeneralException(Exception ex) {
        logger.error("An unexpected error occurred", ex);
        return buildResponse("An internal error occurred. Please contact support.",
HttpStatus.INTERNAL_SERVER_ERROR);
    }

    // Helper method to centralize response creation (SRP)
    private ResponseEntity<ErrorResponse> buildResponse(String message, HttpStatus
status) {
        ErrorResponse error = new ErrorResponse(
```

```
            message,
            status.value(),
            LocalDateTime.now()
        );
        return new ResponseEntity<>(error, status);
    }
}
```

Key Changes Made:

- **Replaced System.err with SLF4J**: This follows best practices for enterprise Java and decouples the output stream from the logic.
- **Centralized Response Building**: The buildResponse helper method ensures that if you ever need to add a timestamp or a "Trace ID" to your errors (as shown in the updated ErrorResponse suggestion), you only change it in one place.
- **Improved Error Messages**: For general exceptions, it hides the raw exception message from the end-user, which is a security best practice, while still logging the full stack trace for developers.
