

## CheckoutController SRP 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.

```
...  
  
@RestController  
@RequestMapping("/api/checkout")  
public class CheckoutController {  
  
    private final CheckoutService checkoutService;  
  
    public CheckoutController(CheckoutService checkoutService) {  
        this.checkoutService = checkoutService;  
    }  
  
    @PostMapping  
    public ResponseEntity checkout(@Valid @RequestBody  
CheckoutRequest request) {  
        RentalAgreement agreement = checkoutService.checkout(  
            request.getToolCode(),  
            request.getRentalDayCount(),  
            request.getDiscountPercent(),  
            request.getCheckoutDate()  
        );  
  
        System.out.println(RentalAgreementFormatter.format(agreement));  
        return ResponseEntity.ok(agreement);  
    }  
  
    @ExceptionHandler(IllegalArgumentException.class)  
    public ResponseEntity  
handleValidationException(IllegalArgumentException ex) {  
        ErrorResponse error = new ErrorResponse(ex.getMessage());  
        return ResponseEntity.badRequest().body(error);  
    }  
  
    @ExceptionHandler(MethodArgumentNotValidException.class)  
    public ResponseEntity  
handleMethodArgumentNotValid(MethodArgumentNotValidExceptio  
n ex) {  
        String message = ex.getBindingResult()  
            .getFieldErrors()  
            .stream()
```

```

        .map(FieldError::getDefaultMessage)
        .collect(Collectors.joining(", "));
    return ResponseEntity.badRequest().body(new
ErrorResponse(message));
}

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

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

```

---

RouteLLM

## SOLID Analysis of CheckoutController

Overall, this controller is **well-structured**, but there are a couple of improvements worth considering.

---

### What's Already Good

- **SRP** (mostly): The controller handles HTTP routing and delegates business logic to CheckoutService — good separation.
  - **DIP**: Depends on CheckoutService abstraction via constructor injection.
  - **OCP/LSP/ISP**: Not directly applicable here, and no violations present.
- 

### Issues Found

## 1. SRP Violation — ErrorResponse as a nested static class

The ErrorResponse class is defined *inside* the controller. It's a data transfer object (DTO) and has no reason to be coupled to CheckoutController. If another controller needs to return an error response, it can't reuse this without referencing CheckoutController.ErrorResponse.

**Fix:** Move ErrorResponse to its own file.

```
// ErrorResponse.java
@Getter
@Setter
public class ErrorResponse {
    private String message;

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

---

## 2. SRP Violation — Exception handlers belong in a global handler

@ExceptionHandler methods inside a controller only apply to *that* controller. As the app grows, every controller would need to duplicate these handlers. This is a cross-cutting concern and should be centralized.

**Fix:** Move exception handlers to a @RestControllerAdvice class.

```
// GlobalExceptionHandler.java
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<ErrorResponse>
    handleValidationException(IllegalArgumentException ex) {
        return ResponseEntity.badRequest().body(new ErrorResponse(ex.getMessage()));
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse>
    handleMethodArgumentNotValid(MethodArgumentNotValidException ex) {
        String message = ex.getBindingResult()
```

```
        .getFieldErrors()
        .stream()
        .map(FieldError::getDefaultMessage)
        .collect(Collectors.joining(", "));
    return ResponseEntity.badRequest().body(new ErrorResponse(message));
}
}
```

---

### 3. Minor — System.out.println in a controller

Using `System.out.println` is not production-appropriate and mixes logging concerns into the controller.

**Fix:** Replace with a proper logger.

```
private static final Logger log = LoggerFactory.getLogger(CheckoutController.class);

// then in the method:
log.info(RentalAgreementFormatter.format(agreement));
```

---

## Refactored CheckoutController

After applying all fixes, the controller becomes clean and focused solely on routing:

```
@RestController
@RequestMapping("/api/checkout")
public class CheckoutController {

    private static final Logger log = LoggerFactory.getLogger(CheckoutController.class);
    private final CheckoutService checkoutService;

    public CheckoutController(CheckoutService checkoutService) {
        this.checkoutService = checkoutService;
    }

    @PostMapping
    public ResponseEntity<RentalAgreement> checkout(@Valid @RequestBody
```

```
CheckoutRequest request) {  
    RentalAgreement agreement = checkoutService.checkout(  
        request.getToolCode(),  
        request.getRentalDayCount(),  
        request.getDiscountPercent(),  
        request.getCheckoutDate()  
    );  
    log.info(RentalAgreementFormatter.format(agreement));  
    return ResponseEntity.ok(agreement);  
}  
}
```

---

## Summary

Issue	Principle Violated	Fix
ErrorResponse nested in controller	SRP	Extract to its own DTO class
Exception handlers in controller	SRP	Move to @RestControllerAdvice
System.out.println	SRP (logging concern)	Use Logger

---

**This is good. For this project we will be using system out as a logger. That may not be production ready, but this is a simple demo project and the specification wants to use console output.**

**I want to add some robust logging to the controller and exception handlers. at minimum I want to log when a request is received and the response returned.**

---

## RouteLLM

Since we are sticking with System.out for this demonstration to match the specification's console output requirements, we can still apply **SRP (Single Responsibility Principle)** by ensuring the controller doesn't get cluttered with string concatenation and logging logic.

I recommend moving the exception handling to a `@RestControllerAdvice`. This keeps your controller "thin" and ensures that even validation errors are logged consistently.

## 1. The Refactored Controller

The controller now focuses strictly on the "Happy Path," logging the entry and exit of the request.

```
@RestController
@RequestMapping("/api/checkout")
public class CheckoutController {

    private final CheckoutService checkoutService;

    public CheckoutController(CheckoutService checkoutService) {
        this.checkoutService = checkoutService;
    }

    @PostMapping
    public ResponseEntity<RentalAgreement> checkout(@Valid @RequestBody
    CheckoutRequest request) {
        System.out.println("Received Checkout Request: " + request);

        RentalAgreement agreement = checkoutService.checkout(
            request.getToolCode(),
            request.getRentalDayCount(),
            request.getDiscountPercent(),
            request.getCheckoutDate()
        );

        // Specification requirement: Output the formatted agreement to console
        System.out.println("Checkout Successful. Generated Agreement:");
        System.out.println(RentalAgreementFormatter.format(agreement));

        return ResponseEntity.ok(agreement);
    }
}
```

## 2. The Global Exception Handler

By moving the handlers here, we satisfy **SRP** and provide a central place to log every failure that occurs during the checkout process.

```
@RestControllerAdvice
public class GlobalExceptionHandler {

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

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse>
    handleMethodArgumentNotValid(MethodArgumentNotValidException 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<ErrorResponse> 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."));
    }
}
```

### 3. The Error DTO

Move this to its own file (e.g., ErrorResponse.java) to allow for reuse and cleaner project structure.

```
@Getter
@Setter
public class ErrorResponse {
    private String message;
    private long timestamp;
```

```

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

```

## Key Improvements Made:

- **Separation of Concerns:** The CheckoutController no longer cares *how* errors are handled or formatted; it only cares about processing valid checkouts.
- **Robustness:** Added a catch-all Exception.class handler to ensure that even unexpected runtime errors are logged to the console instead of just returning a generic 500 page with no context.
- **Observability:** Every request now leaves a "breadcrumb" in the console:
  1. Received Checkout Request...
  2. Checkout Successful... (or Checkout Validation Failed...)
  3. The final formatted RentalAgreement.

---

**With all these new files show me an example file structure?**

---

## RouteLLM

For a standard Spring Boot project following **SOLID** principles and the **Toolstore specification**, your file structure should separate concerns into distinct packages. This ensures that your DTOs (Data Transfer Objects), Controllers, and Services don't become tightly coupled.

## Recommended Project Structure

```

src/main/java/com/toolstore
├── Application.java

```

```

# Spring Boot Entry Point
|
├── controller

```

```

# Web Layer (SRP: Routing only)
|   └── CheckoutController.java
|

```



```

├── exception

# Cross-cutting concerns (SRP: Error handling)
|   ├── GlobalExceptionHandler.java
|   └── ErrorResponse.java
|
├── model

# Domain Objects
|   ├── Tool.java
|   ├── ToolType.java
|   └── RentalAgreement.java
|
└── dto

# Data Transfer Objects (Request/Response)
|   └── CheckoutRequest.java
|
└── service

# Business Logic (SRP: Calculations/Rules)
|   └── CheckoutService.java
|
└── util

# Helpers (SRP: Formatting/Static logic)
    └── RentalAgreementFormatter.java

```

## Why this structure follows SOLID:

### 1. Single Responsibility Principle (SRP)

- **controller**: Only handles HTTP mapping and basic request logging. It doesn't know how to calculate a discount or how to format an error message.
- **exception**: Centralizes all error logic. If you add a new controller later, it automatically gets the same logging and error formatting without changing a single line of code.
- **dto**: Separates the data coming *in* from the API (CheckoutRequest) from the internal domain models (Tool).

## 2. Open/Closed Principle (OCP)

- By using a `GlobalExceptionHandler`, you can add new exception types (e.g., `ToolNotFoundException`) by simply adding a new method to that class, rather than modifying the `CheckoutController`.

## 3. Dependency Inversion Principle (DIP)

- The `CheckoutController` depends on the `CheckoutService`. In a more complex version of this app, you would define `CheckoutService` as an **interface**, allowing you to swap implementations (e.g., a `MockCheckoutService` for testing) without changing the controller.

## 4. Interface Segregation Principle (ISP)

- By separating `CheckoutRequest` (input) from `RentalAgreement` (output), you ensure that the client only sees the data they need for each specific action, rather than one giant "ToolData" object that contains everything.

---

With this new exception handling This test fails. It needs to be modified with the new exception behavior.

```
void
checkout_RentalDaysLessThanOne_Returns400WithMessage()
throws Exception {
    // Specification test 2 - modified to be invalid
    CheckoutRequest request = new CheckoutRequest("LADW", 0,
    10, LocalDate.of(2020, 7, 2));

    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
    any(LocalDate.class)))
        .thenThrow(new IllegalArgumentException("Rental day
    count must be 1 or greater"));

    mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isBadRequest())
        .andExpect(jsonPath("$.message").value("Rental day
    count must be 1 or greater"));
}
```

---

## RouteLLM

The test is likely failing because your new ErrorResponse DTO now includes a timestamp field, or because the GlobalExceptionHandler is returning a slightly different JSON structure than the test expects.

If you moved ErrorResponse to its own file and added the timestamp as suggested in the previous step, the JSON returned by the API now looks like this:

```
{
  "message": "Rental day count must be 1 or greater",
  "timestamp": 1708334400000
}
```

## Updated Test Case

To fix the test, ensure your mockMvc expectations match the new structure. You should also add an assertion to verify that the error was logged to the console (since that was a requirement).

```
@Test
void checkout_RentalDaysLessThanOne_Returns400WithMessage() throws Exception {
    // 1. Arrange
    // Note: If you are using @Valid on the RequestBody, the validation might
    // trigger before the service is even called.
    CheckoutRequest request = new CheckoutRequest("LADW", 0, 10, LocalDate.of(2020, 7,
2));

    // Mock the service to throw the exception as per the specification
    when(checkoutService.checkout(anyString(), eq(0), anyInt(), any(LocalDate.class)))
        .thenReturn(new IllegalArgumentException("Rental day count must be 1 or greater"));

    // 2. Act & Assert
    mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isBadRequest())
        // Verify the message field exists and matches
        .andExpect(jsonPath("$.message").value("Rental day count must be 1 or greater"))
        // Verify the new timestamp field exists (Robustness check)
        .andExpect(jsonPath("$.timestamp").exists());
}
```

## Why it might still fail (and how to fix it):

### 1. Bean Validation vs. Service Exception

If your CheckoutRequest class has JSR-303 annotations like `@Min(1) private int rentalDayCount;`, the `MethodArgumentNotValidException` will be thrown **before** the `checkoutService.checkout` method is ever called.

If that's the case, the error message might be different (e.g., "must be greater than or equal to 1"). You should check if your `GlobalExceptionHandler` is catching the right exception.

### 2. JSON Path Matchers

If you are using a custom `ErrorResponse` object, ensure the getters are present. Jackson (the JSON library) requires getters to serialize the object into JSON. If `@Getter` (Lombok) or public `String getMessage()` is missing, the `jsonPath("$.message")` will fail because the field won't be in the JSON output.

### 3. Verification of Console Output

Since you wanted robust logging using `System.out`, you can optionally verify the log occurred in your test using a `SystemOutRule` or by capturing the output stream, though for a simple demo, verifying the **400 Bad Request** status and the **JSON body** is usually sufficient to prove the `GlobalExceptionHandler` worked.

---

The test still produces this error

```
``01:21:36.645 [Test worker] INFO
org.hibernate.validator.internal.util.Version -- HV000001: Hibernate
Validator 9.0.1.Final
01:21:36.859 [Test worker] INFO
org.springframework.mock.web.MockServletContext -- Initializing
Spring TestDispatcherServlet "
01:21:36.860 [Test worker] INFO
org.springframework.test.web.servlet.TestDispatcherServlet --
```

Initializing Servlet "

01:21:36.863 [Test worker] INFO

org.springframework.test.web.servlet.TestDispatcherServlet --

Completed initialization in 3 ms

Received Checkout Request:

com.toolstore.demo.dto.CheckoutRequest@37d6dddc

Request processing failed: java.lang.IllegalArgumentException:

Rental day count must be 1 or greater

jakarta.servlet.ServletException: Request processing failed:

java.lang.IllegalArgumentException: Rental day count must be 1 or greater

at

org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1011)

at

org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:903)

at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:649)

at

org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:874)

at

org.springframework.test.web.servlet.TestDispatcherServlet.service(TestDispatcherServlet.java:72)

at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:710)

at

org.springframework.mock.web.MockFilterChain\$ServletFilterProxy.doFilter(MockFilterChain.java:160)

at

org.springframework.mock.web.MockFilterChain.doFilter(MockFilterChain.java:127)

at

org.springframework.test.web.servlet.MockMvc.perform(MockMvc.java:199)

at

com.toolstore.demo.controller.CheckoutControllerTest.checkout\_RentalDaysLessThanOne\_Returns400WithMessage(CheckoutControllerTest.java:169)

Caused by: java.lang.IllegalArgumentException: Rental day count must be 1 or greater

at

com.toolstore.demo.service.CheckoutService.checkout(CheckoutService.java:25)

at

com.toolstore.demo.controller.CheckoutController.checkout(Check

outController.java:30)  
    at java.base/  
jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)  
    at java.base/java.lang.reflect.Method.invoke(Method.java:565)  
    at  
org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:258)  
    at  
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:190)  
    at  
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:117)  
    at  
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:934)  
    at  
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:853)  
    at  
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:86)  
    at  
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:963)  
    at  
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:866)  
    at  
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1003)  
    ... 9 more

Rental day count must be 1 or greater

java.lang.IllegalArgumentException: Rental day count must be 1 or greater

    at  
com.toolstore.demo.service.CheckoutService.checkout(CheckoutService.java:25)  
    at  
com.toolstore.demo.controller.CheckoutController.checkout(Check

outController.java:30)  
    at java.base/  
jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)  
    at java.base/java.lang.reflect.Method.invoke(Method.java:565)  
    at  
org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:258)  
    at  
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:190)  
    at  
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:117)  
    at  
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:934)  
    at  
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:853)  
    at  
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:86)  
    at  
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:963)  
    at  
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:866)  
    at  
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1003)  
    at  
org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:903)  
        at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:649)  
    at  
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:874)  
    at  
org.springframework.test.web.servlet.TestDispatcherServlet.service(TestDispatcherServlet.java:72)  
        at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:710)

```
    at
org.springframework.mock.web.MockFilterChain$ServletFilterProxy.doFilter(MockFilterChain.java:160)
    at
org.springframework.mock.web.MockFilterChain.doFilter(MockFilterChain.java:127)
    at
org.springframework.test.web.servlet.MockMvc.perform(MockMvc.java:199)
    at
com.toolstore.demo.controller.CheckoutControllerTest.checkout_RentalDaysLessThanOne_Returns400WithMessage(CheckoutControllerTest.java:169)
```

OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

```
> Task :test
CheckoutControllerTest >
checkout_RentalDaysLessThanOne_Returns400WithMessage()
FAILED
    jakarta.servlet.ServletException at CheckoutControllerTest.java:
169
    Caused by: java.lang.IllegalArgumentException at
CheckoutControllerTest.java:169
1 test completed, 1 failed
> Task :test FAILED``
```

---

## RouteLLM

This stacktrace tells us something very specific:

- ✓ The controller is being invoked
- ✓ The service throws `IllegalArgumentException`
- ✗ But your `GlobalExceptionHandler` is **NOT** being applied in the test context

That's why the exception bubbles up as a `ServletException` instead of returning a 400 JSON response.

---

## ✓ Why This Is Happening



You are almost certainly using:

```
@WebMvcTest(CheckoutController.class)
```

When you use `@WebMvcTest`, Spring only loads:

- The controller under test
- MVC infrastructure

It **does NOT automatically load your `@RestControllerAdvice`** unless you explicitly include it.

So your `GlobalExceptionHandler` is not registered in the test slice.

---

## The Fix

You must import the exception handler into the test.

### Option 1 (Recommended)

Add `@Import(GlobalExceptionHandler.class)` to your test class:

```
@WebMvcTest(CheckoutController.class)
@Import(GlobalExceptionHandler.class)
class CheckoutControllerTest {
```

That's it.

This ensures your exception advice is part of the test application context.

---

### Option 2 (Alternative)

If you want to load everything (less isolated):

@SpringBootTest  
@AutoConfigureMockMvc

But for controller unit tests, @WebMvcTest + @Import is cleaner and faster.

---

## Why Your Logging Appears

Notice this line:

Received Checkout Request: com.toolstore.demo.dto.CheckoutRequest@37d6dddc

That proves:

- The controller method ran
- The service threw
- The exception was NOT intercepted by your advice


So the failure confirms it's purely a test configuration issue — not a logic issue.

---

## After Fixing

Your test should now:

- Return HTTP 400
  - Return JSON:

```
json
{
  "message": "Rental day count must be 1 or greater",
  "timestamp": 1708334400000
}
```
  - Pass 
- 

## Bonus Improvement

If you want cleaner test logs, override toString() in CheckoutRequest so console logs are readable:

```

@Override
public String toString() {
    return "CheckoutRequest{" +
        "toolCode=" + toolCode + "\"" +
        ", rentalDayCount=" + rentalDayCount +
        ", discountPercent=" + discountPercent +
        ", checkoutDate=" + checkoutDate +
        "}";
}

```

---

## Final Answer

Add this to your test:

```
@Import(GlobalExceptionHandler.class)
```

Your failure will disappear immediately.

---

**This is my controller test file, all 4 tests that are supposed to trigger a returned 400 error fail with a similar error**

```
```package com.toolstore.demo.controller;
```

```

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import com.toolstore.demo.dto.CheckoutRequest;
import com.toolstore.demo.exception.GlobalExceptionHandler;
import com.toolstore.demo.model.RentalAgreement;
import com.toolstore.demo.service.CheckoutService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.annotation.Import;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

```

```

import
org.springframework.test.web.servlet.setup.MockMvcBuilders;

import java.math.BigDecimal;
import java.time.LocalDate;

import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@ExtendWith(MockitoExtension.class)
@Import(GlobalExceptionHandler.class)
@SpringBootTest
class CheckoutControllerTest {

    private MockMvc mockMvc;
    private ObjectMapper objectMapper;

    @Mock
    private CheckoutService checkoutService;

    @InjectMocks
    private CheckoutController checkoutController;

    @BeforeEach
    void setUp() {
        mockMvc = MockMvcBuilders
            .standaloneSetup(checkoutController)
            .build();

        objectMapper = new ObjectMapper();
        objectMapper.registerModule(new JavaTimeModule());
    }

    // -----
    // Happy Path Tests
    // -----

```

```

@Test
void checkout_ValidRequest_Returns200WithRentalAgreement()
throws Exception {
    // Specification Test 3
    CheckoutRequest request = new CheckoutRequest("LADW", 3,
10, LocalDate.of(2020, 7, 2));
    RentalAgreement mockAgreement =
buildMockAgreement("LADW", "LADDER", "Werner", 3,
        LocalDate.of(2020, 7, 2), LocalDate.of(2020, 7, 5),
        new BigDecimal("1.99"), 2, new BigDecimal("3.98"),
        10, new BigDecimal("0.40"), new BigDecimal("3.58"));

    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
any(LocalDate.class)))
        .thenReturn(mockAgreement);

    mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.toolCode").value("LADW"))
        .andExpect(jsonPath("$.toolType").value("LADDER"))
        .andExpect(jsonPath("$.toolBrand").value("Werner"))
        .andExpect(jsonPath("$.rentalDays").value(3))
        .andExpect(jsonPath("$.chargeDays").value(2))
        .andExpect(jsonPath("$.preDiscountCharge").value(3.98))
        .andExpect(jsonPath("$.discountPercent").value(10))
        .andExpect(jsonPath("$.discountAmount").value(0.40))
        .andExpect(jsonPath("$.finalCharge").value(3.58));

    verify(checkoutService, times(1))
        .checkout("LADW", 3, 10, LocalDate.of(2020, 7, 2));
}

```

```

@Test
void checkout_ZeroDiscount_Returns200() throws Exception {
    // Specification test 4
    CheckoutRequest request = new CheckoutRequest("JAKD", 6,
0, LocalDate.of(2015, 9, 3));
    RentalAgreement mockAgreement =
buildMockAgreement("JAKD", "JACKHAMMER", "DeWalt", 6,
        LocalDate.of(2015, 9, 3), LocalDate.of(2015, 9, 9),
        new BigDecimal("2.99"), 3, new BigDecimal("8.97"),
        0, new BigDecimal("0.00"), new BigDecimal("8.97"));

    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),

```

```

any(LocalDate.class)))
    .thenReturn(mockAgreement);

    mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.toolCode").value("JAKD"))
        .andExpect(jsonPath("$.discountPercent").value(0))
        .andExpect(jsonPath("$.discountAmount").value(0.00))
        .andExpect(jsonPath("$.finalCharge").value(8.97));
}

@Test
void checkout_100PercentDiscount_Returns200() throws
Exception {
    // Specification test 2
    CheckoutRequest request = new CheckoutRequest("LADW", 3,
100, LocalDate.of(2020, 7, 2));
    RentalAgreement mockAgreement =
buildMockAgreement("LADW", "LADDER", "Werner", 3,
    LocalDate.of(2020, 7, 2), LocalDate.of(2020, 7, 5),
    new BigDecimal("1.99"), 2, new BigDecimal("3.98"),
    100, new BigDecimal("3.98"), new BigDecimal("0.00"));

    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
any(LocalDate.class)))
        .thenReturn(mockAgreement);

    mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.discountPercent").value(100))
        .andExpect(jsonPath("$.finalCharge").value(0.00));
}

// -----
// Validation Tests - Service Layer Exceptions
// -----

@Test
void checkout_DiscountOver100_Returns400WithMessage()
throws Exception {
    // Specification test 1
    CheckoutRequest request = new CheckoutRequest("JAKR", 5,

```

```
101, LocalDate.of(2015, 9, 3));
```

```
    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),  
any(LocalDate.class)))  
        .thenThrow(new IllegalArgumentException("Discount  
percent must be between 0 and 100"));
```

```
    mockMvc.perform(post("/api/checkout")  
        .contentType(MediaType.APPLICATION_JSON)  
        .content(objectMapper.writeValueAsString(request)))  
        .andExpect(status().isBadRequest())  
        .andExpect(jsonPath("$.message").value("Discount  
percent must be between 0 and 100"));  
}
```

```
@Test  
void checkout_NegativeDiscount_Returns400WithMessage()  
throws Exception {  
    // Specification test 3 - modified to be invalid  
    CheckoutRequest request = new CheckoutRequest("LADW", 3,  
-1, LocalDate.of(2020, 7, 2));
```

```
    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),  
any(LocalDate.class)))  
        .thenThrow(new IllegalArgumentException("Discount  
percent must be between 0 and 100"));
```

```
    mockMvc.perform(post("/api/checkout")  
        .contentType(MediaType.APPLICATION_JSON)  
        .content(objectMapper.writeValueAsString(request)))  
        .andExpect(status().isBadRequest())  
        .andExpect(jsonPath("$.message").value("Discount  
percent must be between 0 and 100"));  
}
```

```
@Test  
void  
checkout_RentalDaysLessThanOne_Returns400WithMessage()  
throws Exception {  
    // Specification test 2 - modified to be invalid  
    CheckoutRequest request = new CheckoutRequest("LADW", 0,  
10, LocalDate.of(2020, 7, 2));
```

```
    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),  
any(LocalDate.class)))  
        .thenThrow(new IllegalArgumentException("Rental day
```

count must be 1 or greater"));

```
        mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$.message").value("Rental day
count must be 1 or greater"))
            .andExpect(jsonPath("$.timestamp").exists());
    }
```

```
    @Test
    void checkout_InvalidToolCode_Returns400WithMessage()
    throws Exception {
        // Specification test 2 - modified to be invalid
        CheckoutRequest request = new CheckoutRequest("INVALID",
        3, 10, LocalDate.of(2020, 7, 2));

        when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
        any(LocalDate.class)))
            .thenThrow(new IllegalArgumentException("Tool not
        found: INVALID"));
    }
```

```
        mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$.message").value("Tool not found:
INVALID"));
    }
```

```
// -----
// Malformed Request Tests
// -----
```

```
    @Test
    void checkout_MissingToolCode_Returns400() throws Exception
    {
        String requestJson = ""
        {
            "rentalDayCount": 3,
            "discountPercent": 10,
            "checkoutDate": "07/02/20"
        }
        """;
    }
```



```

        MvcResult result = mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(requestJson))
            .andExpect(status().isBadRequest())
            .andReturn();

        MockHttpServletResponse response = result.getResponse();
        System.out.println("Status: " + response.getStatus());
        System.out.println("Error message: " +
response.getErrorMessage());
        System.out.println("Response body: " +
response.getContentAsString());
    }

```

```

@Test
void checkout_MissingCheckoutDate_Returns400() throws
Exception {
    String requestJson = ""
    {
        "toolCode": "LADW",
        "rentalDayCount": 3,
        "discountPercent": 10
    }
    """;

```

```

        MvcResult result = mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(requestJson))
            .andExpect(status().isBadRequest())
            .andReturn();

        MockHttpServletResponse response = result.getResponse();
        System.out.println("Status: " + response.getStatus());
        System.out.println("Error message: " +
response.getErrorMessage());
        System.out.println("Response body: " +
response.getContentAsString());
    }

```

```

@Test
void checkout_EmptyBody_Returns400() throws Exception {
    MvcResult result = mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{}"))
        .andExpect(status().isBadRequest())
        .andReturn();

```

```

        MockHttpServletResponse response = result.getResponse();
        System.out.println("Status: " + response.getStatus());
        System.out.println("Error message: " +
response.getErrorMessage());
        System.out.println("Response body: " +
response.getContentAsString());
    }

```

```

// -----
// Specification Tests
// -----

```

```

@Test
void specification_test_2() throws Exception {
    CheckoutRequest request = new CheckoutRequest("LADW", 3,
10, LocalDate.of(2020, 7, 2));

    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
any(LocalDate.class)))
        .thenReturn(buildMockAgreement("LADW", "LADDER",
"Werner", 3,
            LocalDate.of(2020, 7, 2), LocalDate.of(2020, 7, 5),
            new BigDecimal("1.99"), 2, new BigDecimal("3.98"),
            10, new BigDecimal("0.40"), new
BigDecimal("3.58")));

    mockMvc.perform(post("/api/checkout")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk());

    verify(checkoutService).checkout("LADW", 3, 10,
LocalDate.of(2020, 7, 2));
}

```

```

@Test
void specification_test_3() throws Exception {
    CheckoutRequest request = new CheckoutRequest("CHNS", 5,
25, LocalDate.of(2015, 7, 2));

    when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
any(LocalDate.class)))
        .thenReturn(buildMockAgreement("CHNS", "CHAINSAW",
"Stihl", 5,
            LocalDate.of(2015, 7, 2), LocalDate.of(2015, 7, 7),

```

```
        new BigDecimal("1.49"), 4, new BigDecimal("5.96"),
        25, new BigDecimal("1.49"), new
BigDecimal("4.47")));
```

```
        mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isOk());
```

```
        verify(checkoutService).checkout("CHNS", 5, 25,
        LocalDate.of(2015, 7, 2));
    }
```

**@Test**

```
void specification_test_4() throws Exception {
    CheckoutRequest request = new CheckoutRequest("JAKD", 6,
    0, LocalDate.of(2015, 9, 3));
```

```
        when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
        any(LocalDate.class)))
            .thenReturn(buildMockAgreement("JAKD",
        "JACKHAMMER", "DeWalt", 6,
        LocalDate.of(2015, 9, 3), LocalDate.of(2015, 9, 9),
        new BigDecimal("2.99"), 3, new BigDecimal("8.97"),
        0, new BigDecimal("0.00"), new BigDecimal("8.97")));
```

```
        mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isOk());
```

```
        verify(checkoutService).checkout("JAKD", 6, 0,
        LocalDate.of(2015, 9, 3));
    }
```

**@Test**

```
void specification_test_5() throws Exception {
    CheckoutRequest request = new CheckoutRequest("JAKR", 9,
    0, LocalDate.of(2015, 7, 2));
```

```
        when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
        any(LocalDate.class)))
            .thenReturn(buildMockAgreement("JAKR",
        "JACKHAMMER", "Ridgid", 9,
        LocalDate.of(2015, 7, 2), LocalDate.of(2015, 7, 11),
        new BigDecimal("2.99"), 6, new BigDecimal("17.94"),
```

```

        0, new BigDecimal("0.00"), new
BigDecimal("17.94"))));

        mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isOk());

        verify(checkoutService).checkout("JAKR", 9, 0,
LocalDate.of(2015, 7, 2));
    }

    @Test
    void specification_test_6() throws Exception {
        CheckoutRequest request = new CheckoutRequest("JAKR", 4,
50, LocalDate.of(2020, 7, 2));

        when(checkoutService.checkout(anyString(), anyInt(), anyInt(),
any(LocalDate.class)))
            .thenReturn(buildMockAgreement("JAKR",
"JACKHAMMER", "Ridgid", 4,
                LocalDate.of(2020, 7, 2), LocalDate.of(2020, 7, 6),
                new BigDecimal("2.99"), 1, new BigDecimal("2.99"),
                50, new BigDecimal("1.50"), new
BigDecimal("1.49")));

        mockMvc.perform(post("/api/checkout")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isOk());

        verify(checkoutService).checkout("JAKR", 4, 50,
LocalDate.of(2020, 7, 2));
    }

    // -----
    // Helper
    // -----

    private RentalAgreement buildMockAgreement(
        String toolCode, String toolType, String toolBrand, int
rentalDays,
        LocalDate checkoutDate, LocalDate dueDate, BigDecimal
dailyCharge,
        int chargeDays, BigDecimal preDiscountCharge, int
discountPercent,

```

```

        BigDecimal discountAmount, BigDecimal finalCharge) {

    return RentalAgreement.builder()
        .toolCode(toolCode)
        .toolType(toolType)
        .toolBrand(toolBrand)
        .rentalDays(rentalDays)
        .checkoutDate(checkoutDate)
        .dueDate(dueDate)
        .dailyRentalCharge(dailyCharge)
        .chargeDays(chargeDays)
        .preDiscountCharge(preDiscountCharge)
        .discountPercent(discountPercent)
        .discountAmount(discountAmount)
        .finalCharge(finalCharge)
        .build();
    }
}

```

---

## RouteLLM

Your failures make sense with this test setup: you're using **standalone MockMvc**:

```

mockMvc = MockMvcBuilders
    .standaloneSetup(checkoutController)
    .build();

```

In *standalone* mode, Spring does **not** automatically discover/apply `@RestControllerAdvice` beans from the application context. Also, your `@Import(GlobalExceptionHandler.class)` and `@SpringBootTest` don't help much here because you aren't building `MockMvc` from the `Spring WebApplicationContext`.

So the `IllegalArgumentException` escapes the controller and becomes a `ServletException` (exactly what you're seeing).

## Fix it (keep standalone setup)

Attach your advice explicitly to the standalone `MockMvc` builder:

```

@BeforeEach
void setUp() {

```

```

mockMvc = MockMvcBuilders
    .standaloneSetup(checkoutController)
    .setControllerAdvice(new GlobalExceptionHandler())
    .build();

objectManager = new ObjectMapper();
objectManager.registerModule(new JavaTimeModule());
}

```

That should make all 400-tests that rely on your `GlobalExceptionHandler` pass.

## Why new `GlobalExceptionHandler()` is OK here

Your `GlobalExceptionHandler` (as shown earlier) has no dependencies. If it *does* have dependencies, then instead of `new ...()` you'd need to construct it with mocks, or switch to the Spring context approach below.

---

## Alternative fix (more “Spring-realistic”)

If you want advice, validation, Jackson config, etc. to behave exactly like runtime, don't use standalone. Use Spring's configured `MockMvc`.

## Option A: `@WebMvcTest` (recommended for controller tests)

This is the usual Spring approach.

```

@WebMvcTest(CheckoutController.class)
@Import(GlobalExceptionHandler.class)
class CheckoutControllerTest {

    @Autowired private MockMvc mockMvc;
    @Autowired private ObjectMapper objectMapper;

    @MockBean private CheckoutService checkoutService;
}

```

Then **delete**:

- @ExtendWith(MockitoExtension.class)
- @SpringBootTest
- manual MockMvcBuilders...
- manual ObjectMapper creation

## Option B: @SpringBootTest + @AutoConfigureMockMvc

Loads more of the app (slower, but simplest if you want full integration-ish behavior):

```
@SpringBootTest
@AutoConfigureMockMvc
class CheckoutControllerTest {
    @Autowired MockMvc mockMvc;
    @Autowired ObjectMapper objectMapper;
}
```

And use @MockBean CheckoutService checkoutService;

---

## What you should do in *your* file with minimal change

Given you already have a bunch of tests and they currently use standaloneSetup, the smallest change is:

1. **Remove** @Import(GlobalExceptionHandler.class) (it's not used by standalone)
2. **Add** .setControllerAdvice(new GlobalExceptionHandler())

So:

```
@BeforeEach
void setUp() {
    mockMvc = MockMvcBuilders
        .standaloneSetup(checkoutController)
        .setControllerAdvice(new GlobalExceptionHandler())
        .build();

    objectMapper = new ObjectMapper();
    objectMapper.registerModule(new JavaTimeModule());
}
```

That will stop the exceptions from bubbling up and will return your JSON error payload with HTTP 400.

---

## One more note about your “Malformed Request Tests”

Those missing-field / bad JSON tests often rely on:

- Jackson deserialization errors (`HttpMessageNotReadableException`)
- validation errors (`MethodArgumentNotValidException`)

If your `GlobalExceptionHandler` doesn't handle those exception types (or you changed behavior), those tests might start failing too. If you want, paste your current `GlobalExceptionHandler` and I'll tell you exactly which additional handlers you should add so those malformed request tests return consistent 400s with the same `{ "message": "..."} structure.`

---