**Title: Liskov Substitution Principle (LSP) in Laravel - Payment Gateway Example**

**Introduction to Liskov Substitution Principle (LSP)**

* **Definition:** Subclasses should be able to replace their superclass without altering the correctness of the program.
* **Goal:** Ensure that objects of derived classes can be used interchangeably with objects of the base class.
* **Key Benefit:** Prevents unexpected behavior when switching between different implementations.

**Violating LSP: A Bad Example**

**Problem: Bank Transfer Throws an Exception**

class PaymentGateway {

public function processPayment(float $amount) {

return "Processing payment of $$amount.";

}

}

class BankTransferPayment extends PaymentGateway {

public function processPayment(float $amount) {

throw new Exception("Bank transfer requires manual approval.");

}

}

**Why does this break LSP?**

* ❌ PaymentGateway assumes that all subclasses can process payments directly.
* ❌ BankTransferPayment unexpectedly throws an exception instead of processing a payment.
* ❌ If BankTransferPayment is used in place of PaymentGateway, it **breaks the application**.

**Fixing LSP: Using an Interface and Proper Handling**

**Step 1: Introduce an Interface**

interface PaymentGatewayInterface {

public function processPayment(float $amount): string;

}

✅ Defines a contract ensuring that all payment methods implement processPayment().

**Step 2: Implement Payment Methods Correctly**

class PayPalPayment implements PaymentGatewayInterface {

public function processPayment(float $amount): string {

return "Paid $$amount via PayPal.";

}

}

class BankTransferPayment implements PaymentGatewayInterface {

public function processPayment(float $amount): string {

return "Bank transfer of $$amount initiated. Awaiting approval.";

}

}

✅ No unexpected exceptions. ✅ Consistent behavior across different payment methods.

**Where Should the Exception Be Handled?**

The exception should be handled **before** processing, not inside the processPayment() method.

**Step 3: Implement a Validation Layer**

class PaymentValidator {

public static function validate(PaymentGatewayInterface $paymentGateway) {

if ($paymentGateway instanceof BankTransferPayment) {

throw new Exception("Bank transfers must be approved manually before processing.");

}

}

}

✅ Ensures validation happens **before** executing payment. ✅ Keeps processPayment() behavior **consistent**.

**Step 4: Use Validation in Controller**

public function processPayment(Request $request, PaymentGatewayInterface $paymentGateway) {

$amount = $request->input('amount', 100);

try {

PaymentValidator::validate($paymentGateway);

$result = $paymentGateway->processPayment($amount);

return response()->json(['message' => $result]);

} catch (Exception $e) {

return response()->json(['error' => $e->getMessage()], 400);

}

}

✅ Ensures BankTransferPayment follows the same behavior contract as other payments. ✅ Prevents unexpected exceptions **before processing the payment**.

**Final Takeaways**

| **❌ Bad Approach (Violates LSP)** | **✅ Good Approach (Follows LSP)** |
| --- | --- |
| Throws an exception **inside processPayment()** | Returns a **status message** instead of breaking execution |
| Unexpected behavior for subclasses | Consistent response for all payment methods |
| Code crashes when switching to BankTransferPayment | Validation is handled **separately** before processing |
| Violates expectations of PaymentGatewayInterface | Respects contract while still enforcing rules |

✅ **LSP ensures reliability:** Substituting a subclass should not break the program. ✅ **Use interfaces, not rigid base classes:** This allows different implementations to behave correctly. ✅ **Ensure every subclass follows the contract:** No unexpected exceptions or behavior changes.

**Q&A Summary**

**Q: Where should the exception for bank transfers be implemented?**  
**A:** It should be handled **before** processing the payment, in a **validation layer**, rather than inside the processPayment() method.

**Q: Why does throwing an exception inside processPayment() break LSP?**  
**A:** Because it changes the expected behavior of the method, making BankTransferPayment unusable as a direct substitute for other payment methods.

**Q: How does using an interface solve the problem?**  
**A:** It ensures that all payment methods follow the same contract while allowing specific rules (like manual approval) to be handled **outside** the core payment logic.

🎯 **This document serves as a structured reference for understanding LSP in Laravel’s Payment Gateway system!** 🚀