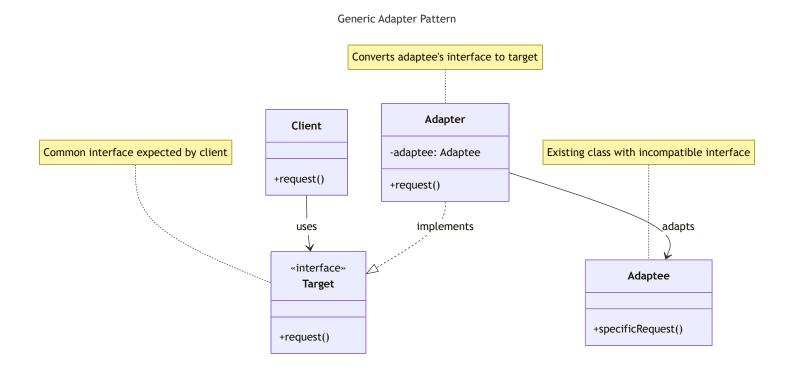
Generic Diagram



Adapter Pattern for Insurance Providers Integration

Without Adapter Pattern

• Principle of least Knowledge

VIOLATES MULTIPLE SOLID PRINCIPLES - Tightly coupled to all APIs - Must change for new providers- Multiple responsibilities InsuranceClient +processTravelGuardClaim() +processAutoProtectClaim() +getTravelGuardStatus() : String +getAutoProtectStatus() : String -convertAutoProtectEnum() : String +addNewProvider(): // Requires modification directly coupled directly coupled handles conversion «enumeration» TravelGuardApi AutoProtectApi AutoProtectStatus **APPROVED** PENDING +submitClaim(String, Double) +addClaim(Double) REJECTED +getClaimStatus(String) : String +getStatus(String) : AutoProtectStatus

```
// BAD - Without Adapter Pattern
class InsuranceClient {
    private TravelGuardApi travelGuard = new TravelGuardApi();
    private AutoProtectApi autoProtect = new AutoProtectApi();
    public void submitClaim(String provider, String id, Double amount) {
        if ("TravelGuard".equals(provider)) {
            travelGuard.submitClaim(id, amount);
        } else if ("AutoProtect".equals(provider)) {
            autoProtect.addClaim(amount); // Different method name!
        }
        // Add new provider = modify this method X
    }
    public String getStatus(String provider, String id) {
        if ("TravelGuard".equals(provider)) {
            return travelGuard.getClaimStatus(id);
        } else if ("AutoProtect".equals(provider)) {
            return convertEnum(autoProtect.getStatus(id)); // Manual conversion X
        }
        return null;
    }
}
```

SOLID Principles Violated Without Adapter Pattern

Single Responsibility Principle (SRP)

- Client handles business logic + API conversions + provider-specific logic
- · One class doing multiple unrelated tasks

Open/Closed Principle (OCP)

- Must modify client code to add new insurance providers
- Not open for extension, requires modification

Dependency Inversion Principle (DIP)

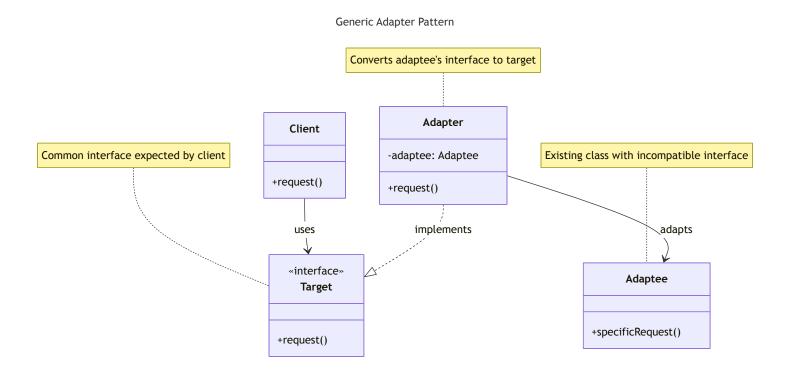
- Client directly depends on concrete API classes
- High-level modules depending on low-level modules

Interface Segregation Principle (ISP)

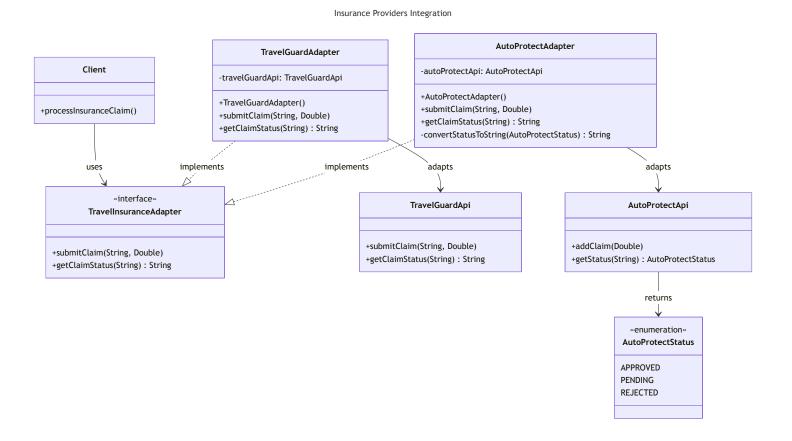
- · Client forced to know about all provider-specific methods
- · No common interface to work with

With Adapter Pattern

Generic Diagram



Specific Diagram



How Adapter Pattern Helps Here

Key Benefits:

- Unified Interface: All insurance providers accessed through same TravelInsuranceAdapter interface
- Easy Integration: Add new providers without changing existing code
- API Isolation: External API changes only affect their specific adapter
- Data Format Standardization: Converts different return types (enum → string) to common format
- Parameter Mapping: Handles different method signatures seamlessly
- Maintainability: Client code remains unchanged when switching providers