Structural Pattern: Adapter

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

Describing the adapter pattern

- Class adapter
- Object adapter

Structure of the adapter pattern

Implementation

- Loading cities from an external system for a travel agent application

Coming Up

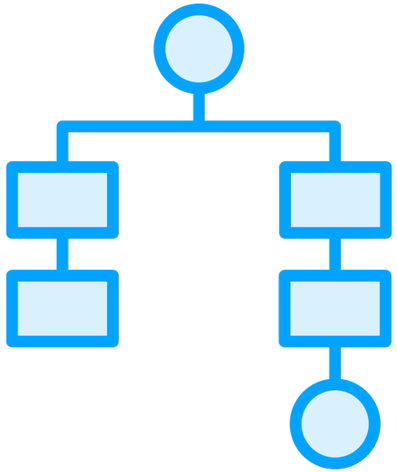
Use cases for this pattern

Pattern consequences

Related patterns

Describing the Adapter Pattern

Creational



Structural

Behavioral

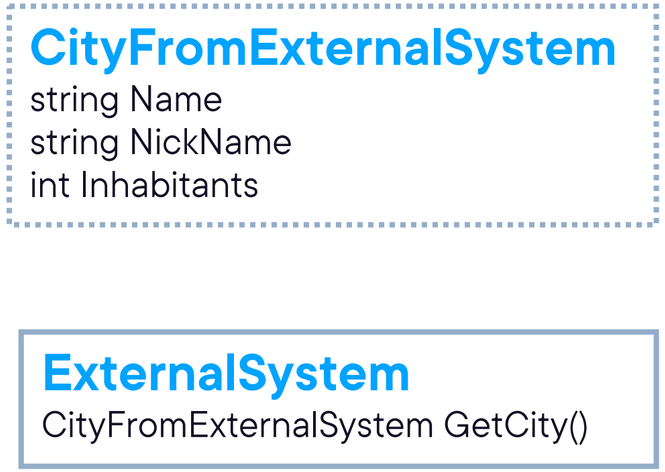
Adapter

The intent of this pattern is to convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

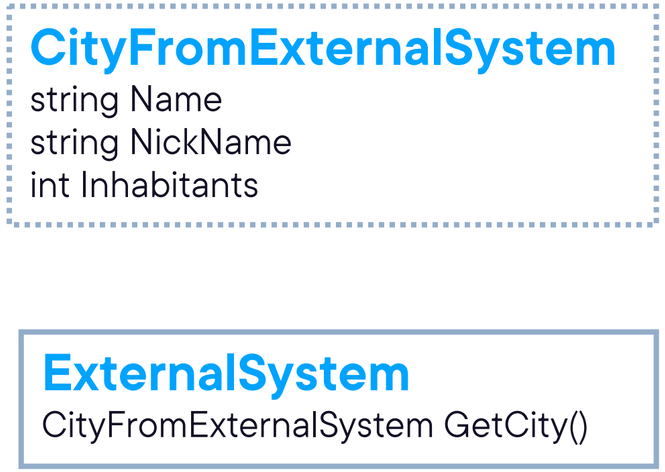
Describing the Object Adapter Pattern

CityFromExternalSystem
string Name
string NickName
int Inhabitants

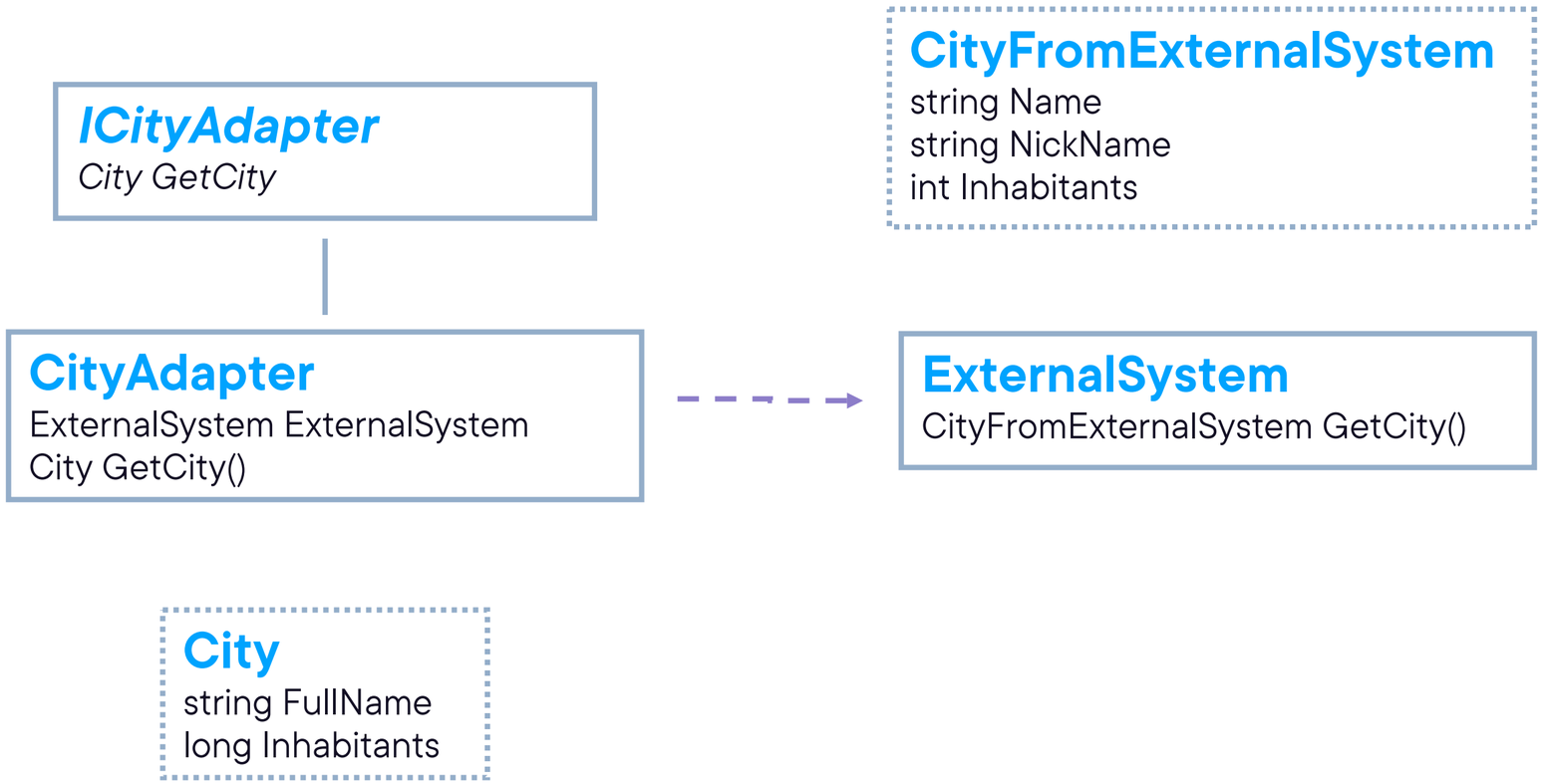
Describing the Object Adapter Pattern



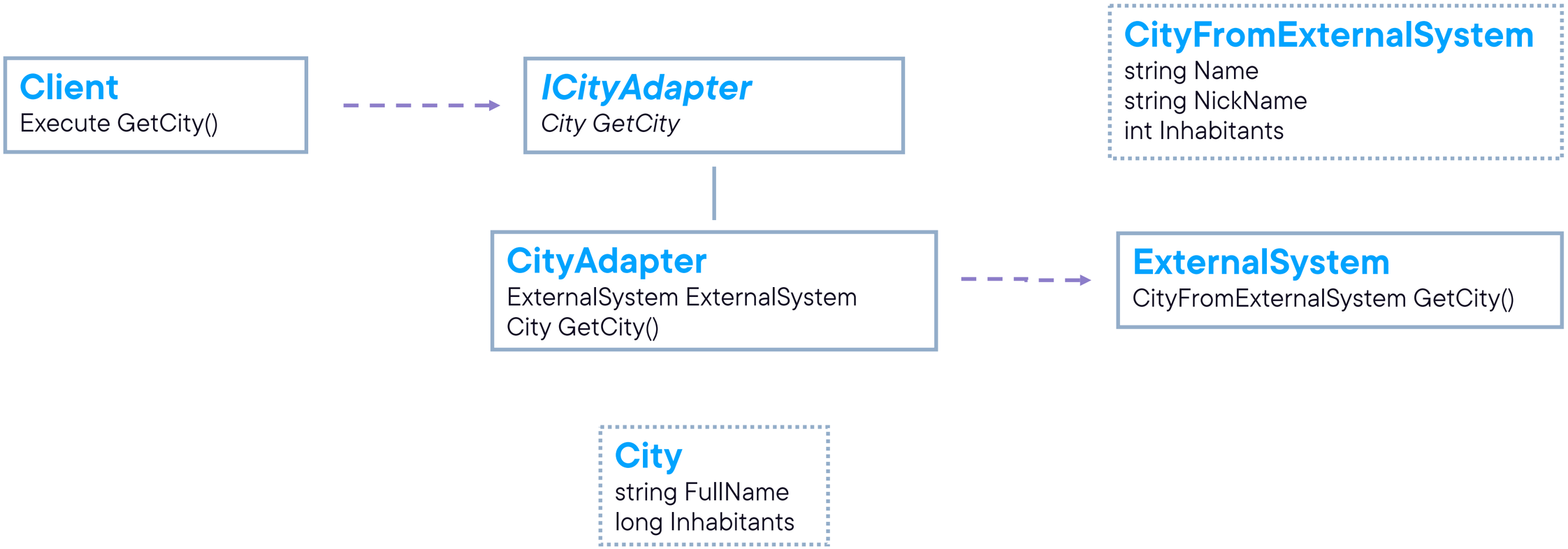
Describing the Object Adapter Pattern



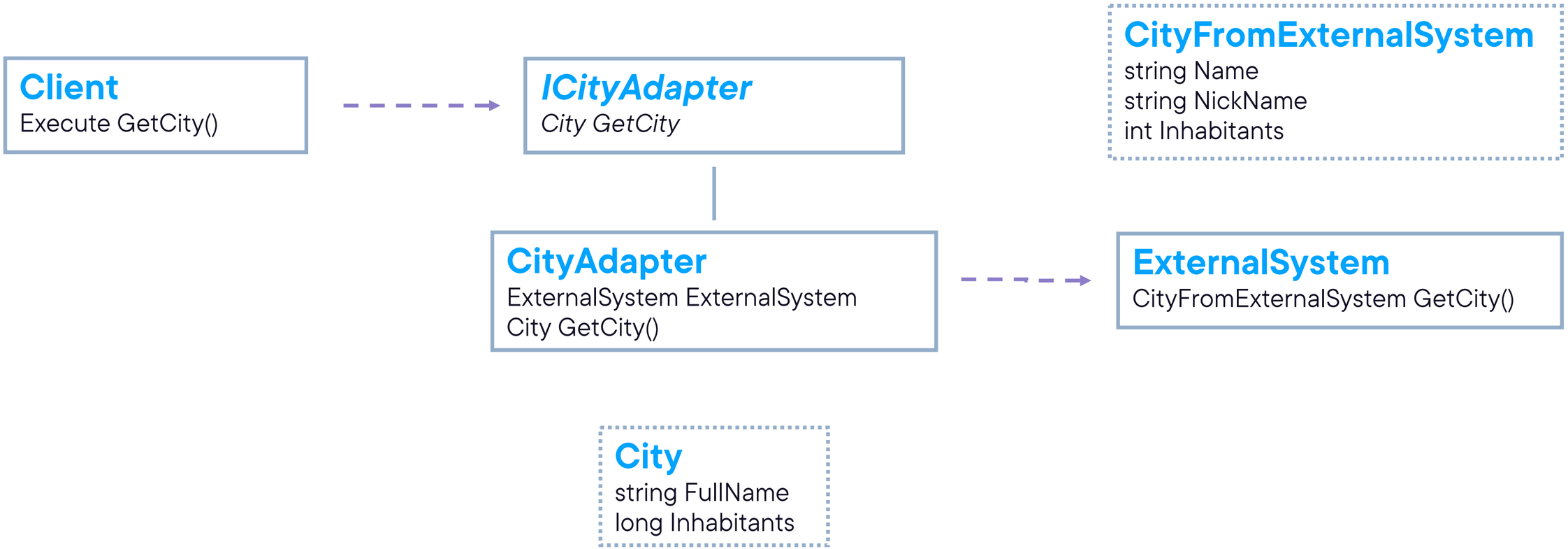
Describing the Object Adapter Pattern



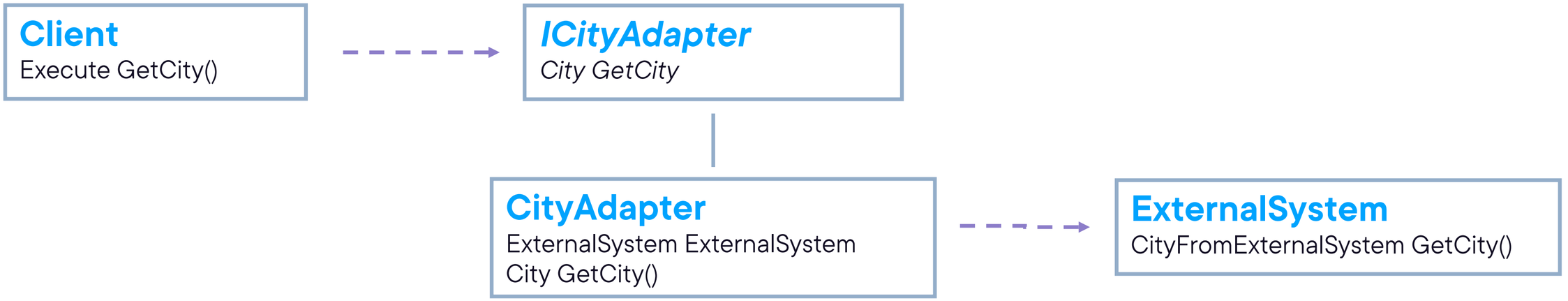
Describing the Object Adapter Pattern



Object Adapter Pattern Structure

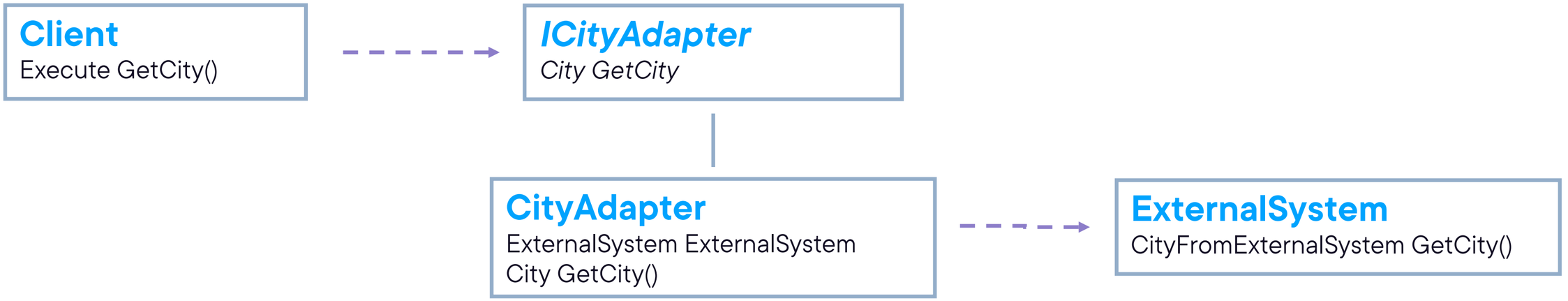


Object Adapter Pattern Structure

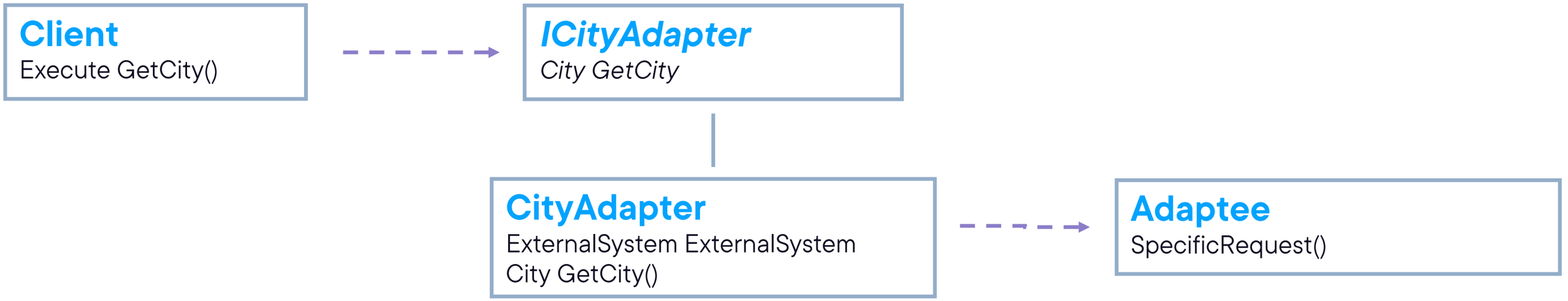


`Adaptee` **defines an existing interface that needs to be adapted**

Object Adapter Pattern Structure

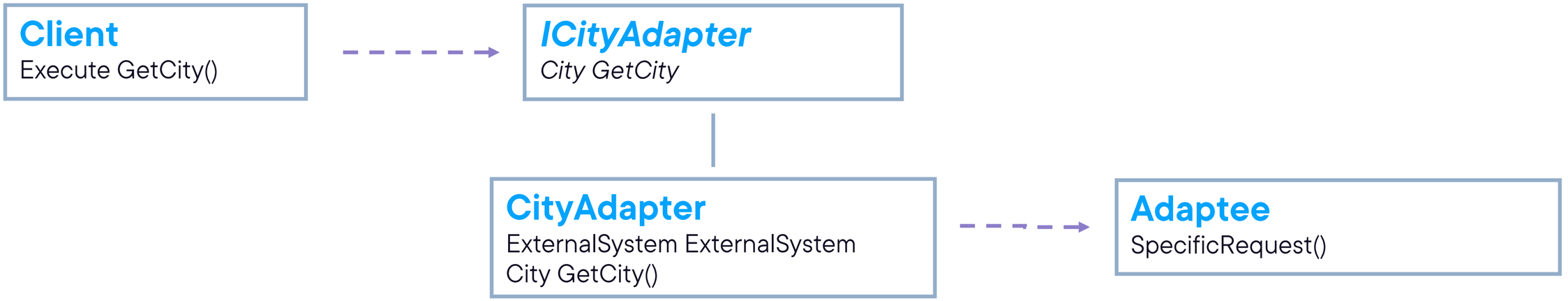


Object Adapter Pattern Structure

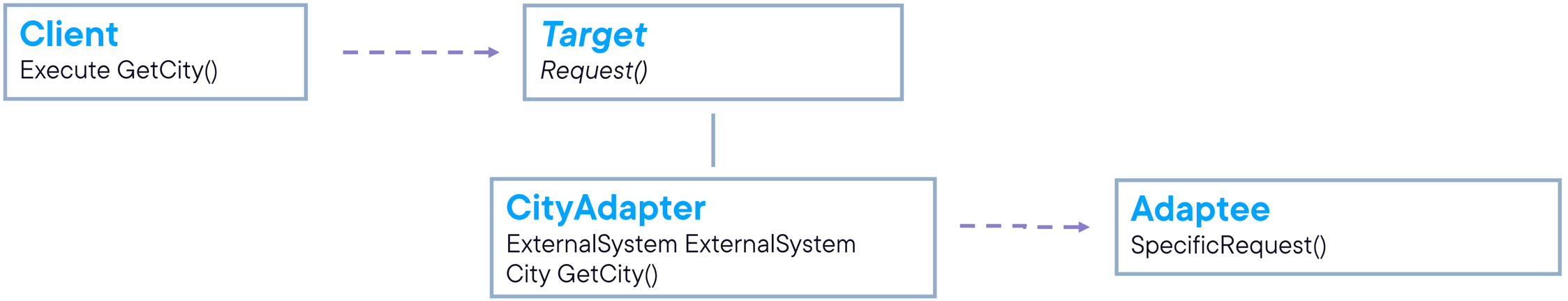


`Target` **defines the domain-specific interface that the** `Client` **uses**

Object Adapter Pattern Structure

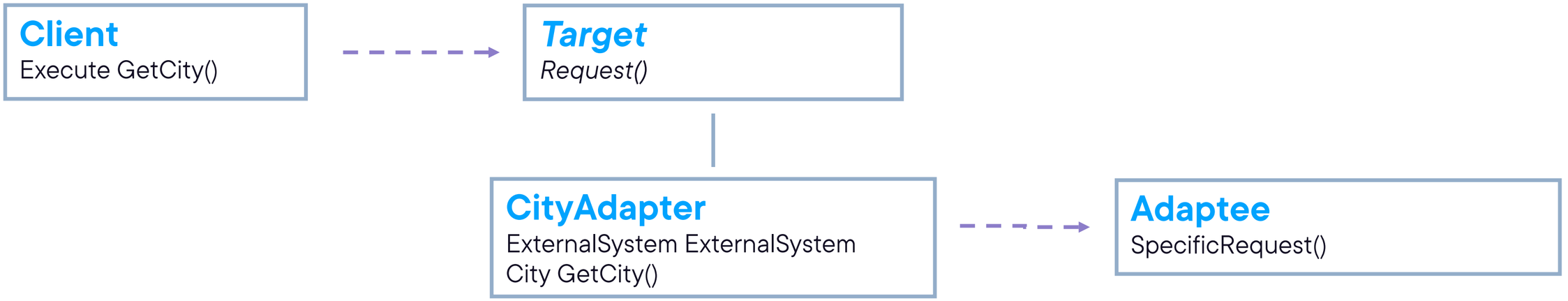


Object Adapter Pattern Structure

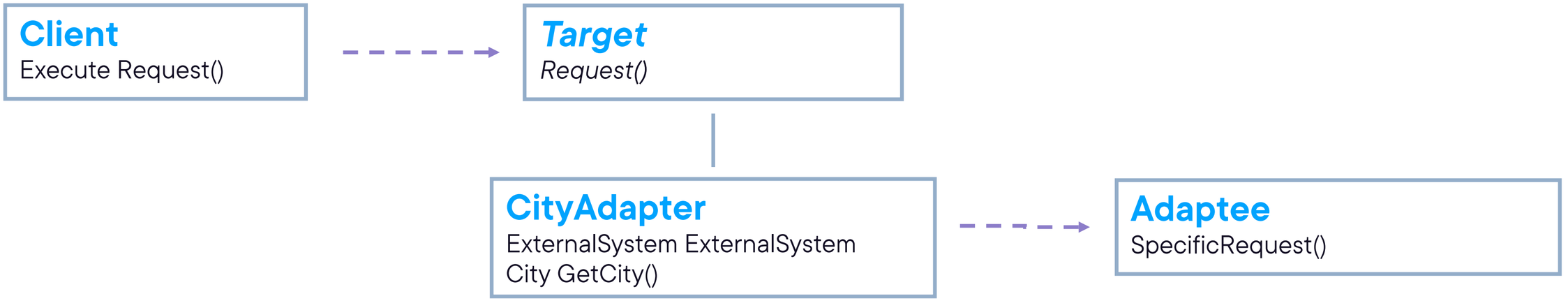


`Client` **collaborates with objects conforming to the** `Target` **interface**

Object Adapter Pattern Structure

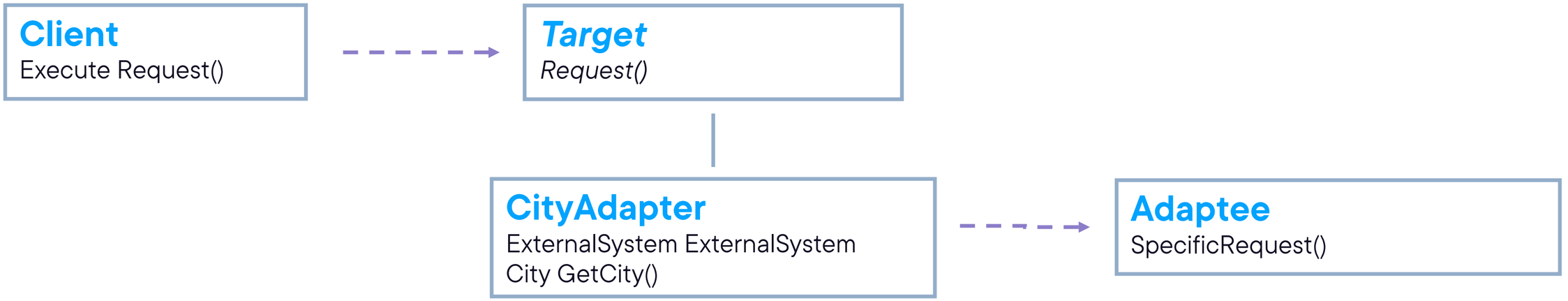


Object Adapter Pattern Structure

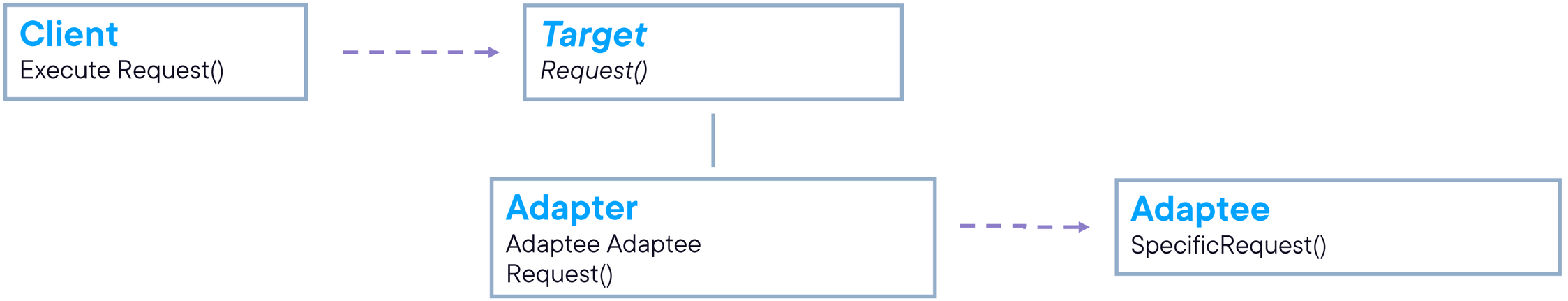


Adapter **adapts the interface of the** Adaptee **to the** Target **interface**

Object Adapter Pattern Structure



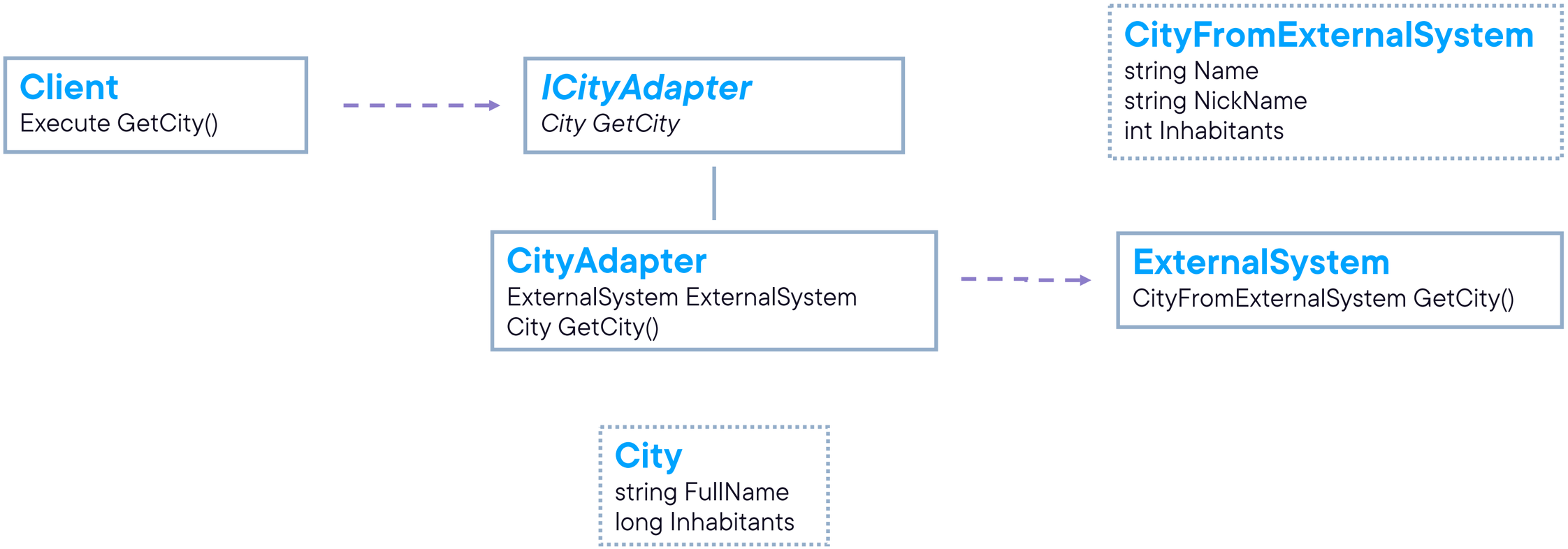
Object Adapter Pattern Structure



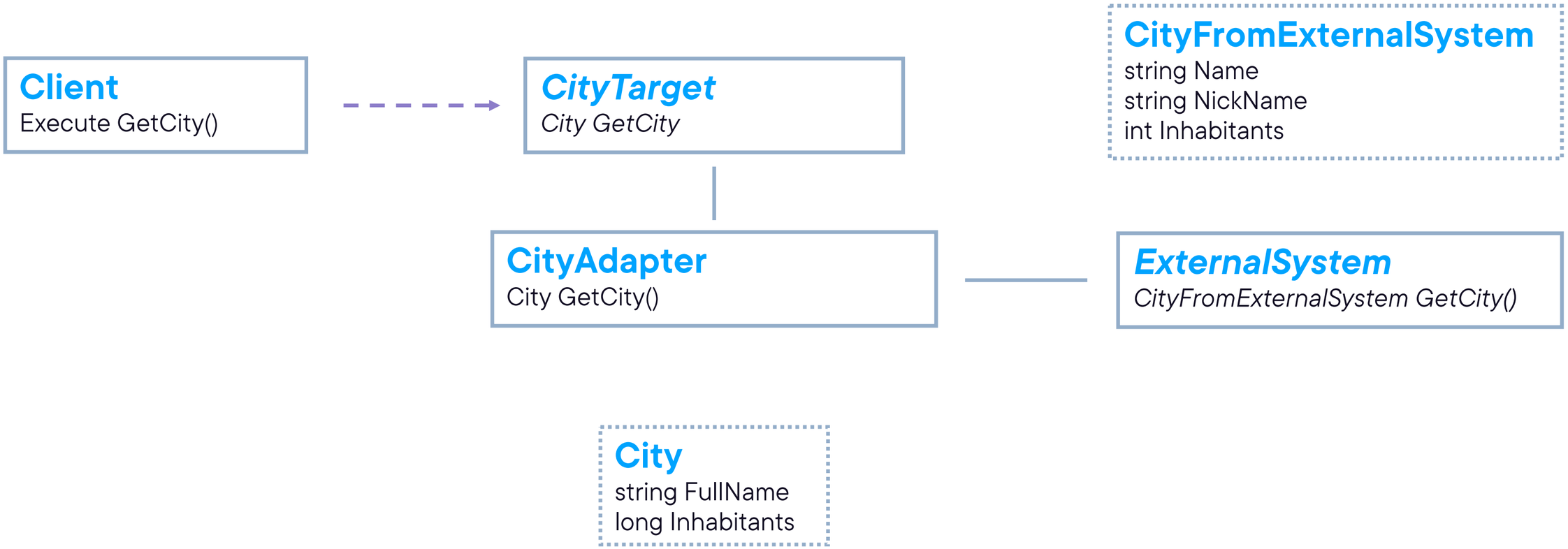
Demo

Implementing the object adapter pattern

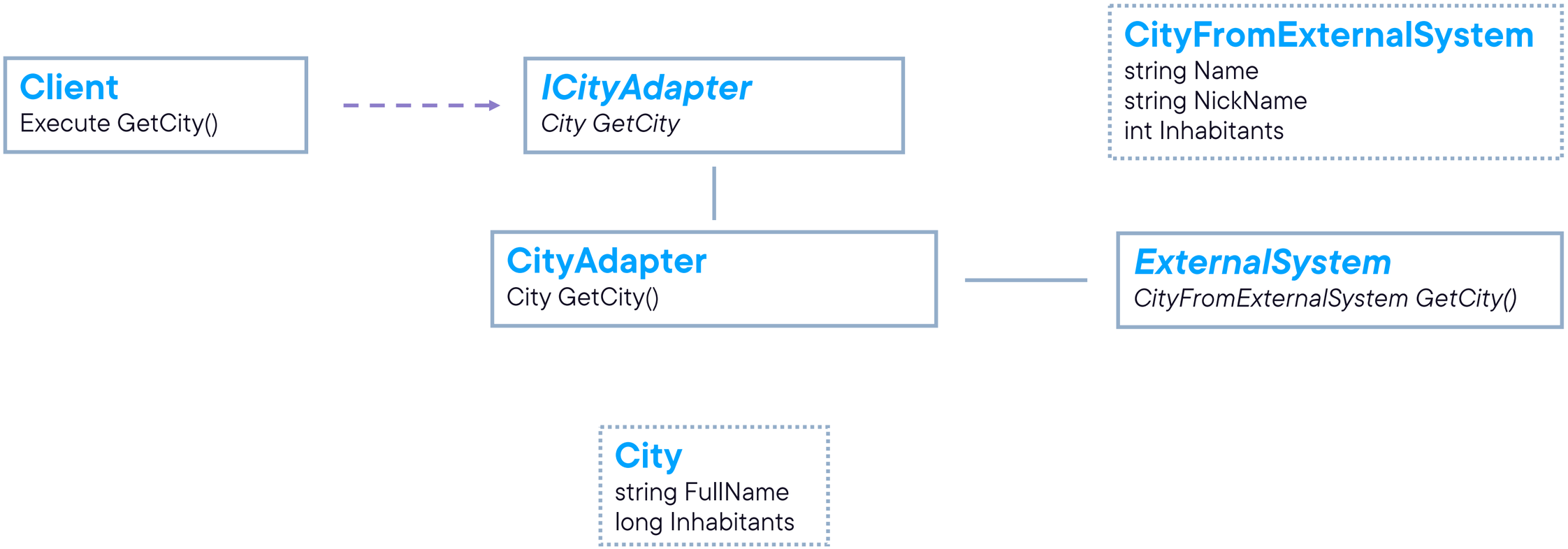
Describing the Class Adapter Pattern



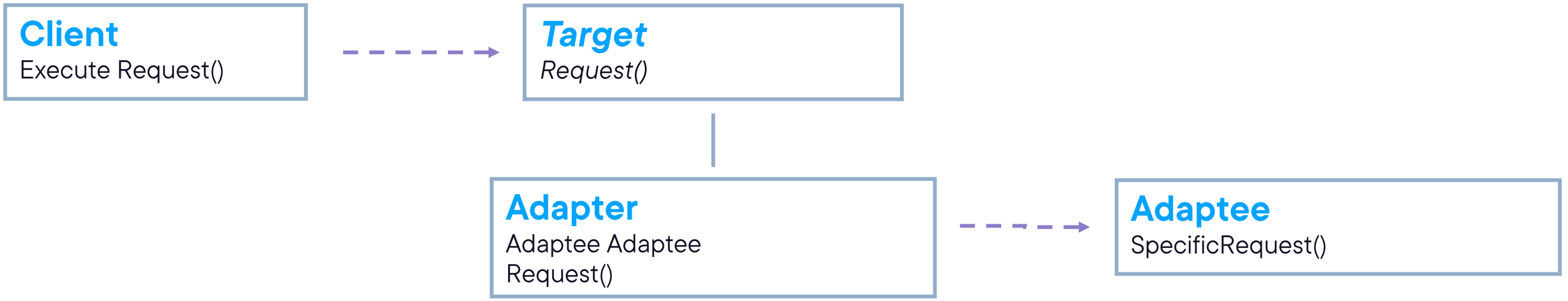
Describing the Class Adapter Pattern



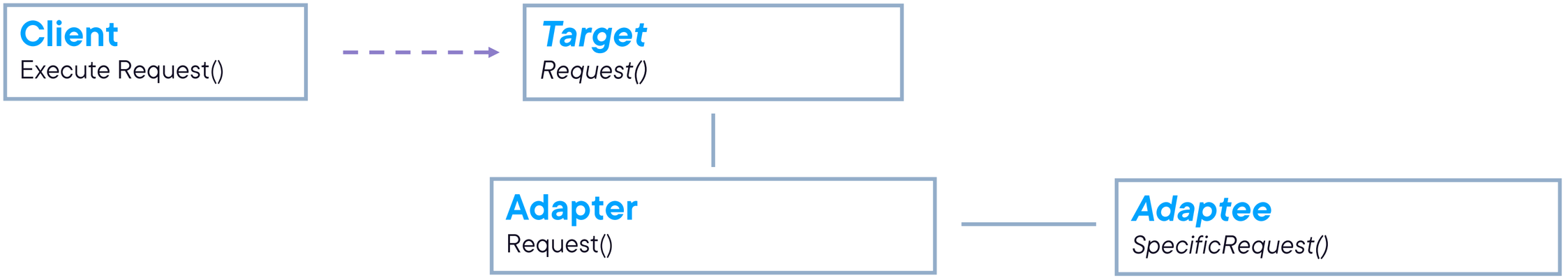
Describing the Class Adapter Pattern



Class Adapter Pattern Structure



Class Adapter Pattern Structure



Demo

Implementing the class adapter pattern

Use Cases for the Adapter Pattern

When you want to use an existing class but the interface does not match the one you need

When you want to create a reusable class (the adapter) that works with classes that don't have compatible interfaces

When you need to use several existing subclasses, don't want create additional subclasses for each of them, but still need to adapt their interface

Use Cases for the Adapter Pattern

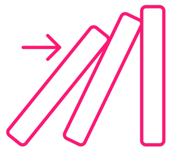
Integrating with a third-party library

Integrating with a web service

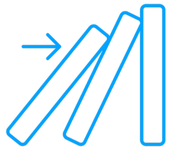
Mocking objects during testing

Integrating with logging or monitoring frameworks

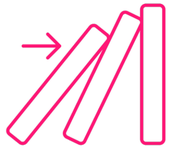
Pattern Consequences



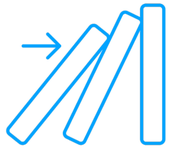
A single adapter can work with many adaptees, and can add functionality to all adaptees at once



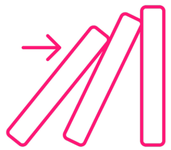
The interface (adapter code) is separated out from the rest of the code: single responsibility principle



New types of adapters can be introduced without breaking client code: open/closed principle

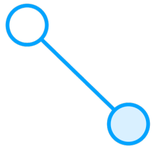


The object adapter makes it hard to override adaptee behavior

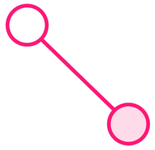


Additional complexity is introduced

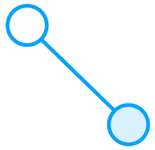
Related Patterns



Bridge
Bridge separates interface from implementation, adapter changes the interface of an existing object



Decorator
Decorator changes an object without changing its interface, adapter changes the interface of an existing object



Facade
With façade you define a new interface for an entire subsystem, with adapter you're making an existing interface useable via wrapping

Proxy
Defines a surrogate for another object, but does not change its interface

Summary

Intent of the adapter pattern:

- Let classes work together that couldn't before because of incompatible interfaces

Summary

Two types:

- Object adapter relies on composition
- Class adapter relies on multiple inheritance

Up Next:

Structural Pattern: Bridge