Design Patterns

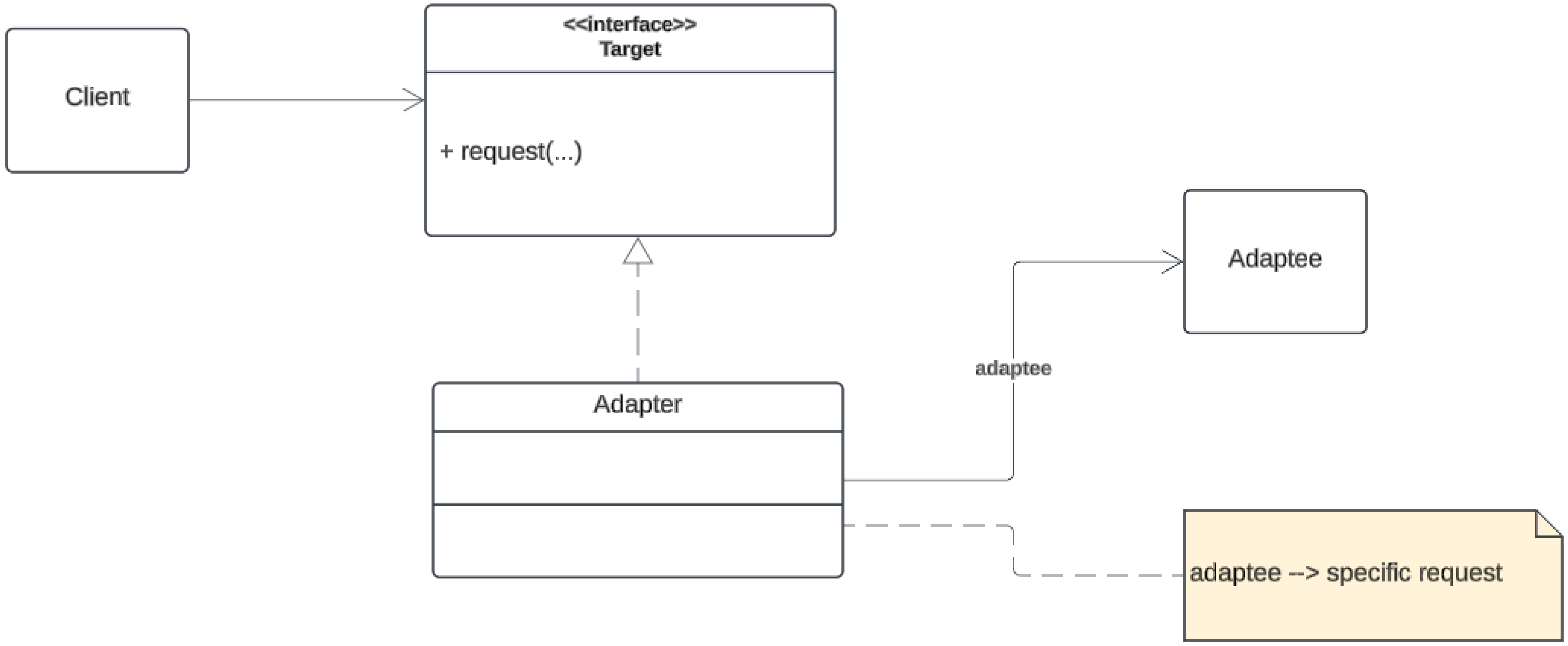
Adapter pattern

Dr. Chad Williams
Central Connecticut State University

Design pattern: Adapter

- Structural pattern
- Motivation:
  - Toolkit (often 3rd party) isn't reusable only because interface doesn't match domain specific interface
  - Interact with one language encapsulate interaction with different type of language code related to that component is contained
- Example:
  - Wrapping C++ library for calling from Java
  - Wrapping System calls to appear like normal method calls

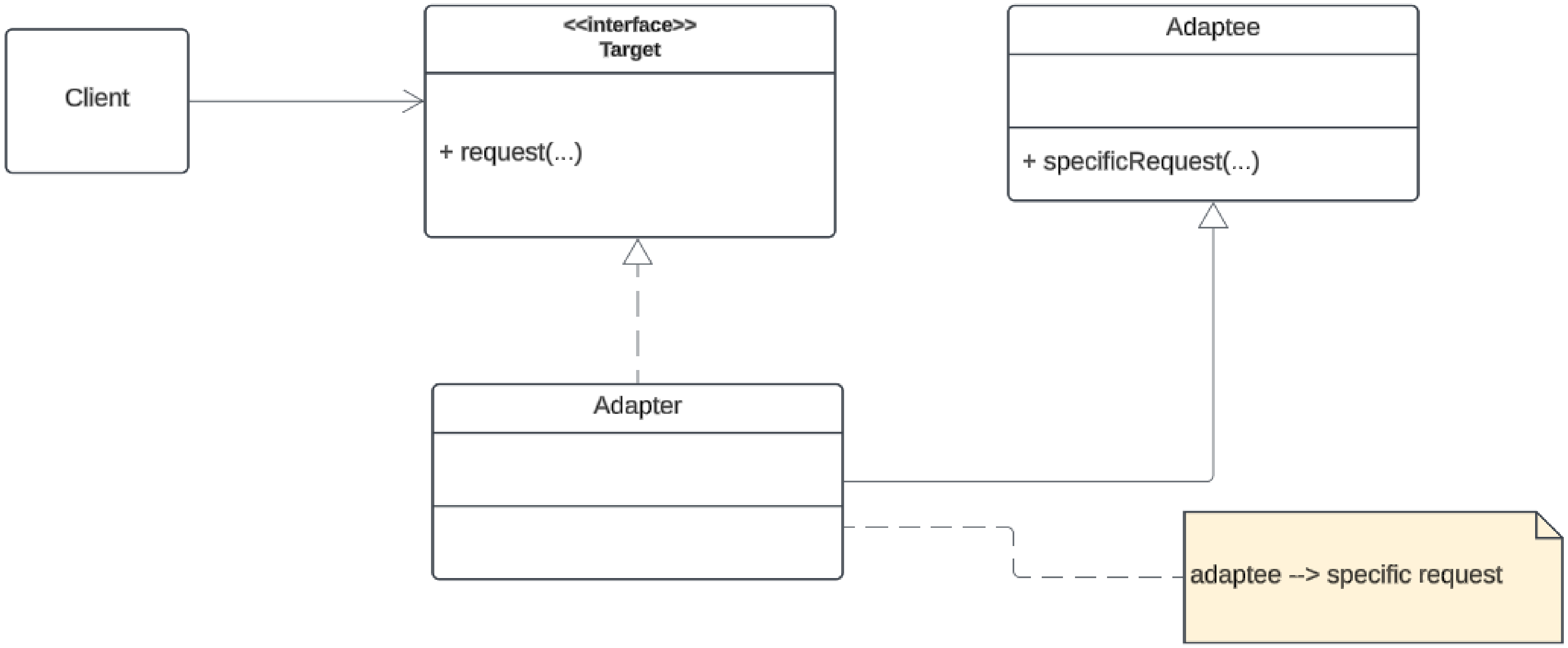
Object Adapter UML – object composition



Note the association to the Adaptee could be aggregation if needs to be held or could be a dependency.

Typical for calls to outside packages, languages or targets (system), adapter translates how to call Adaptee

Class Adapter UML

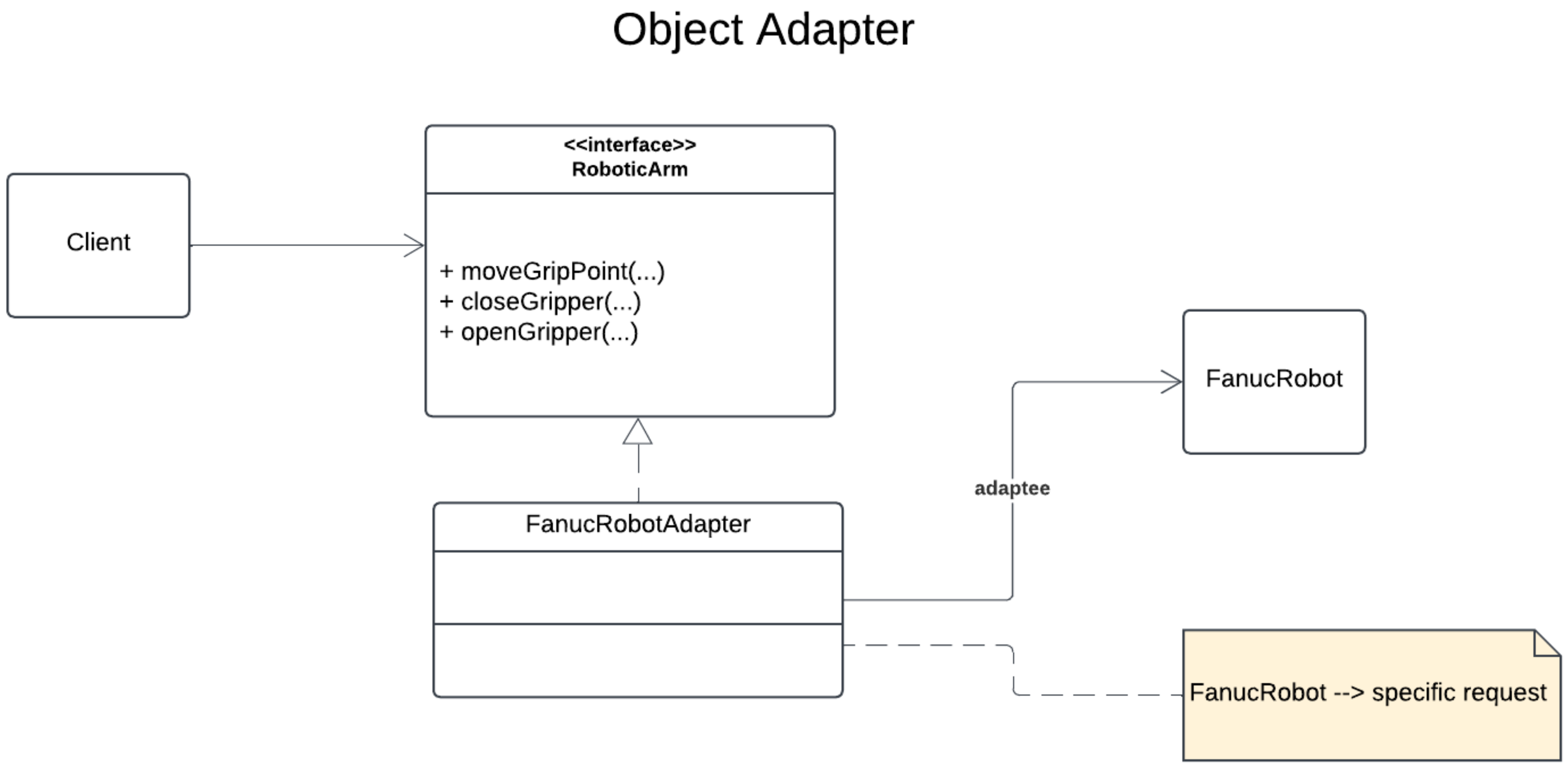


Requires Adapter to extend Adaptee and provide common interface

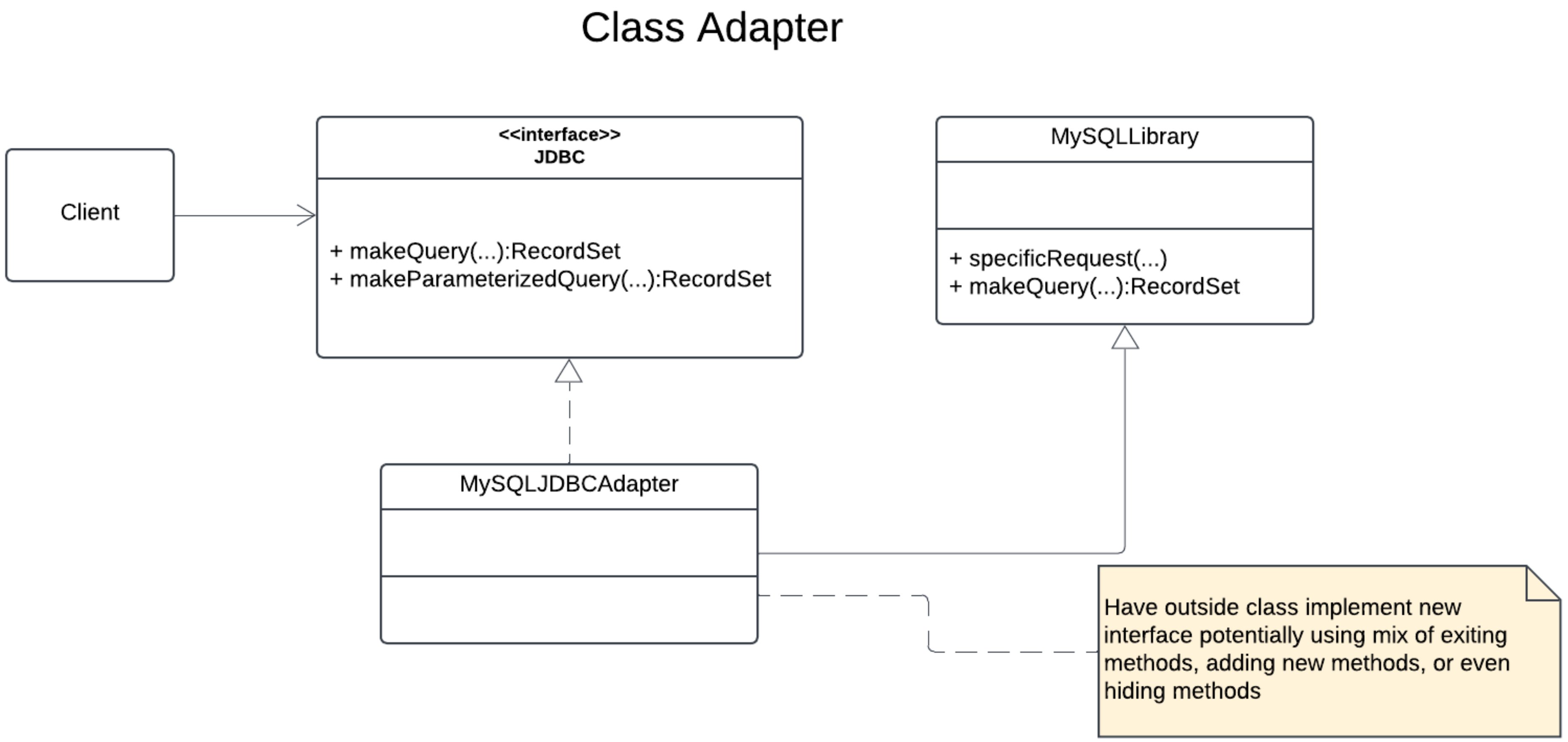
Considerations

- Class Adapter
  - Commits to concrete Adaptee class, as a consequence a class adapter won't work when want to adapt a class AND all of its subclasses
  - Easy to override just some behavior
  - Can be used to restrict interface to prevent dependencies on proprietary methods
- Object adapter
  - Lets single Adapter potentially work with multiple Adaptees (Adaptee and all subclasses)
  - Makes it harder to override Adaptee behavior

Sample structure



Sample structure


Class Adapter
Client
<<interface>>
JDBC
+ makeQuery(...):RecordSet
+ makeParameterizedQuery(...):RecordSet
MySQLLibrary
+ specificRequest(...)
+ makeQuery(...):RecordSet
MySQLJDBCAdapter
Have outside class implement new interface potentially using mix of exiting methods, adding new methods, or even hiding methods

Relevant patterns

- Bridge – separate intent, separate interface from implementation
- Decorator – enhance object without changing interface
- Proxy – Defines representative or surrogate for another object and does not change its interface