Design Patterns & Software Architecture
Adapter

dr. Joost Schalken-Pinkster

Windesheim University of Applied Science

The Netherlands

The contents of these course slides is (in great part) based on:
Chris Loftus, *Course on Design Patterns & Software Architecture for NEU*. Aberystwyth University, 2013.
Jeroen Weber & Christian Köppe, *Course on Patterns and Frameworks*. Hogeschool Utrecht, 2013.
Leo Pruijt, *Course on Software Architecture*. Hogeschool Utrecht, 2010-2013.

Session overview

- Adapter

Adapter design pattern

Let’s find a design pattern

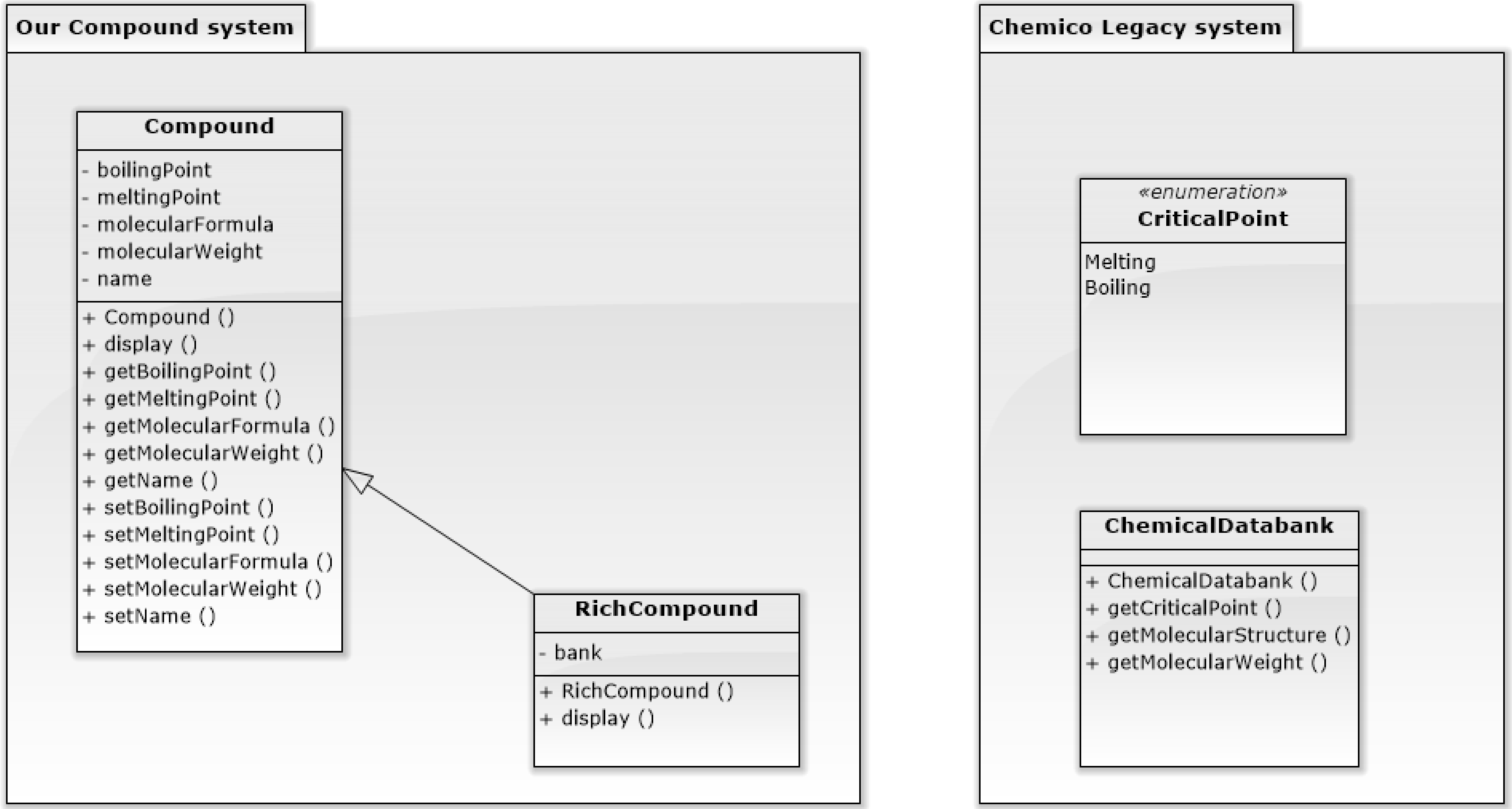
Will now present, on the board,
and using Eclipse,
a solution that utilises
the adapter design pattern…

Case: Chemical compounds database Requirements

You have been asked to maintain an application that provides information on chemical compounds. Currently, the application has a Compound class that maintains the following information: Compound name, boiling point, melting point, molecular weight and formula.

You receive a maintenance requirement:

- Integrate our existing application with a legacy application used by Chemico, a company we have just acquired. In particular, they have an extensive chemical information bank…
- Make sure that client software that uses our application is not adversely affected by this integration

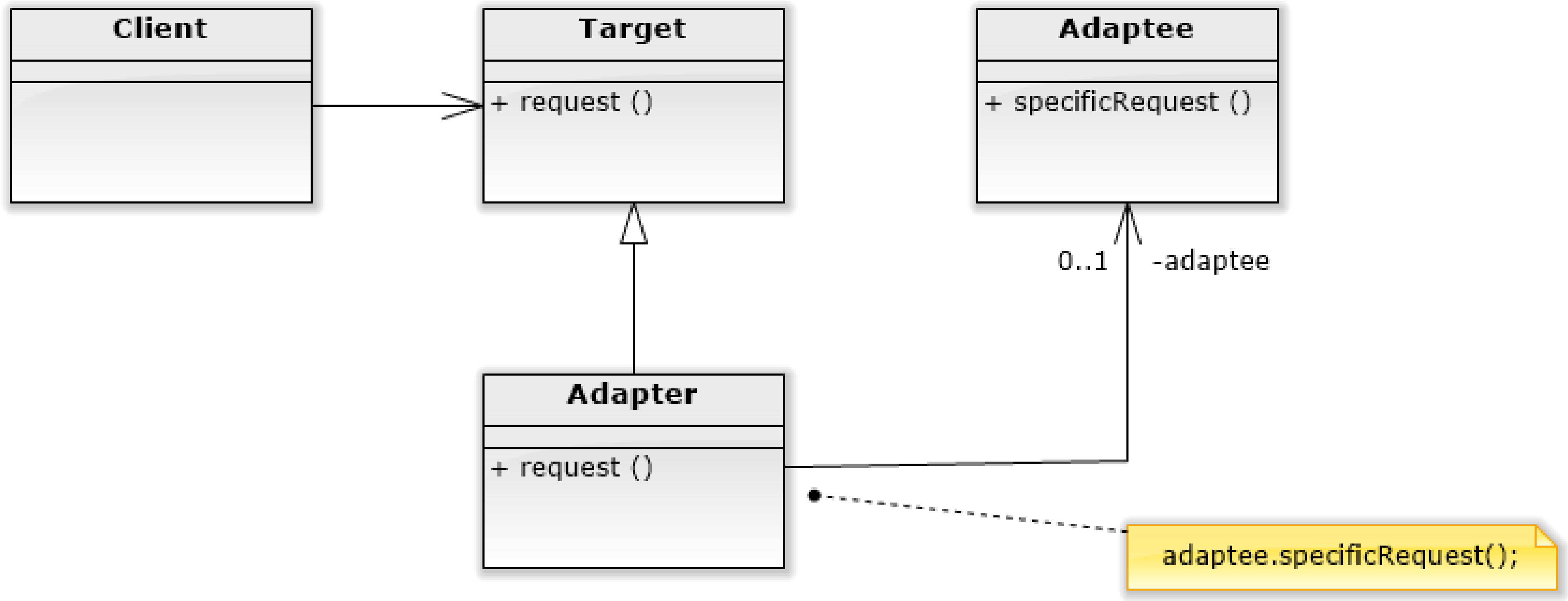

Our Compound system
Compound
- boilingPoint
- meltingPoint
- molecularFormula
- molecularWeight
- name
+ Compound ()
+ display ()
+ getBoilingPoint ()
+ getMeltingPoint ()
+ getMolecularFormula ()
+ getMolecularWeight ()
+ getName ()
+ setBoilingPoint ()
+ setMeltingPoint ()
+ setMolecularFormula ()
+ setMolecularWeight ()
+ setName ()
RichCompound
- bank
+ RichCompound ()
+ display ()
Chemico Legacy system
«enumeration»
CriticalPoint
Melting
Boiling
ChemicalDatabank
+ ChemicalDatabank ()
+ getCriticalPoint ()
+ getMolecularStructure ()
+ getMolecularWeight ()

Adapter pattern definition

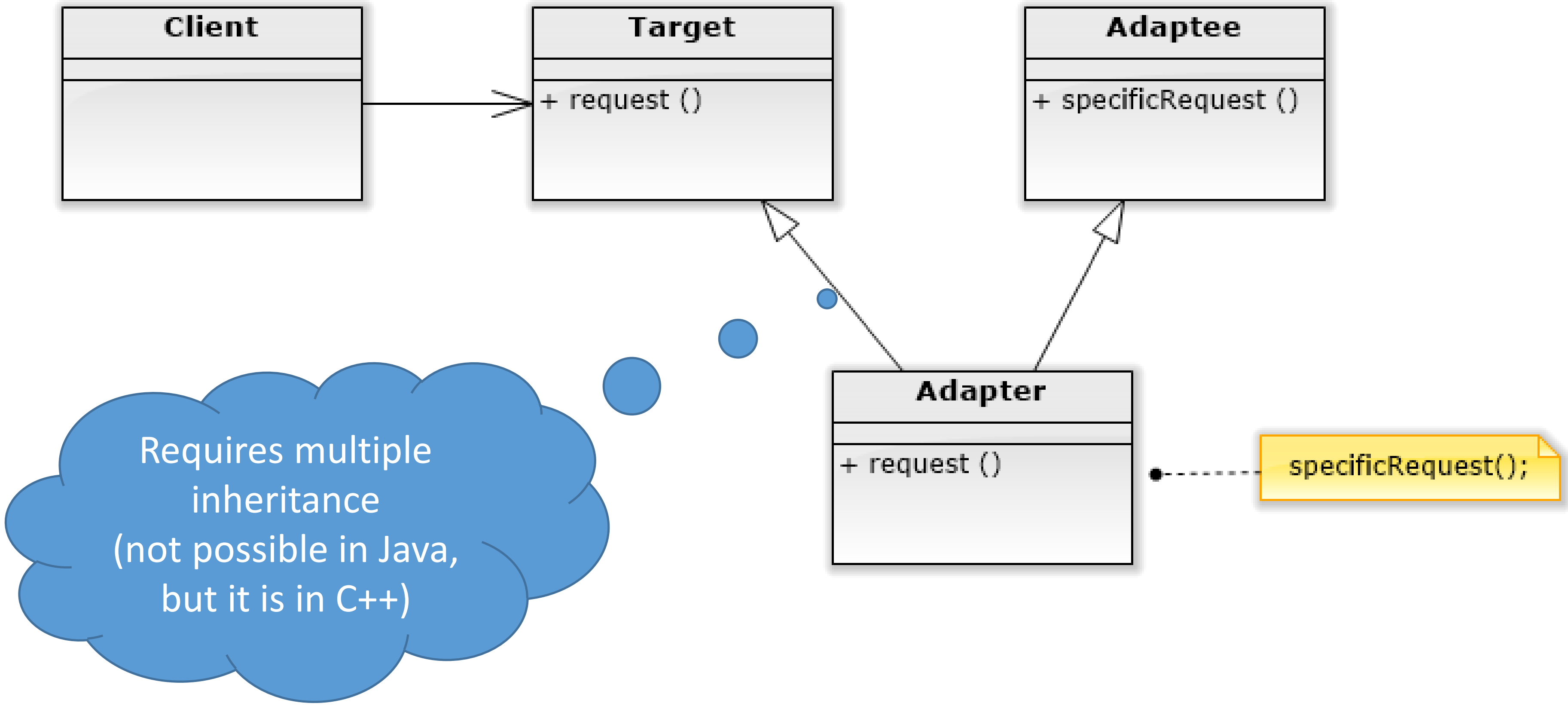
- **Intent**: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.
- **Motivation**: You have an application that stores information about chemical compounds. At some point you want to reuse and integrate an existing chemical databank application. But you must not break your client code. You decide to use an adapter…

- **Applicability**: Use the pattern when:
  - You want to use an existing class, and its interface does not match the one you need.

Structure (Object adapter):



Structure (Class adapter):



- **Participants:**
  - Target
    - defines the domain-specific interface that Client uses.
  - Client
    - collaborates with objects conforming to the Target interface.
  - Adaptee
    - defines an existing interface that needs adapting.
  - Adapter
    - adapts the interface of Adaptee to the Target interface.

- **Consequences:** The class adapter pattern
  - Helps to make classes more reusable…
- **Implementation:**
  - What instance of the adaptee class should be called?…
  - The amount of adaptation will increase as the difference between the expected and actual class interfaces increases…

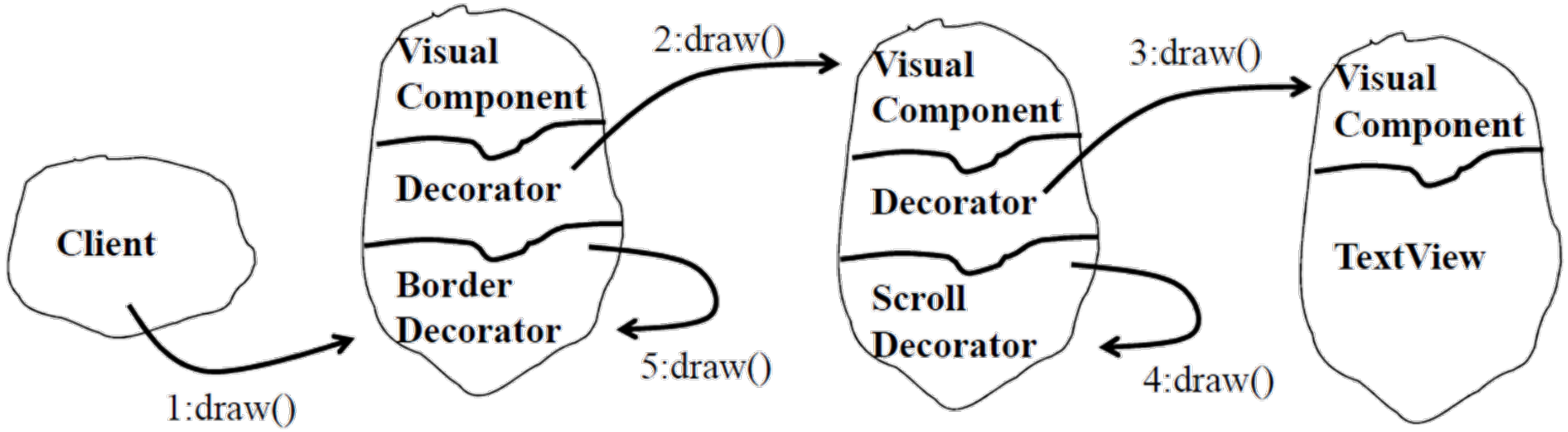
Class and object adapters have different trade-offs (consequences).

- **An object adapter**
  - lets a single Adapter work with many Adaptees—that is, the Adaptee itself and all of its subclasses (if any). The Adapter can also add functionality to all Adaptees at once.
  - makes it harder to override Adaptee behavior. It will require subclassing Adaptee and making Adapter refer to the subclass rather than the Adaptee itself.

- **A class adapter**
  - adapts Adaptee to Target by committing to a concrete Adapter class. As a consequence, a class adapter won't work when we want to adapt a class *and* all its subclasses.
  - lets Adapter override some of Adaptee's behavior, since Adapter is a subclass of Adaptee.
  - introduces only one object, and no additional pointer indirection is needed to get to the Adaptee.

How does Adapter differ from Decorator?

- In decorators, all decorators and concrete components share a common super-type:



- In Adapter this is **cannot** be the case: the target and Adaptee classes were developed separately, so no common super-type.

Reading

For this lesson please read:

- Chapter 7 (Being adaptive) of Head First Design Patterns until page 254 (And now for something completely different).