SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

27 Design Patterns

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 15, 2018

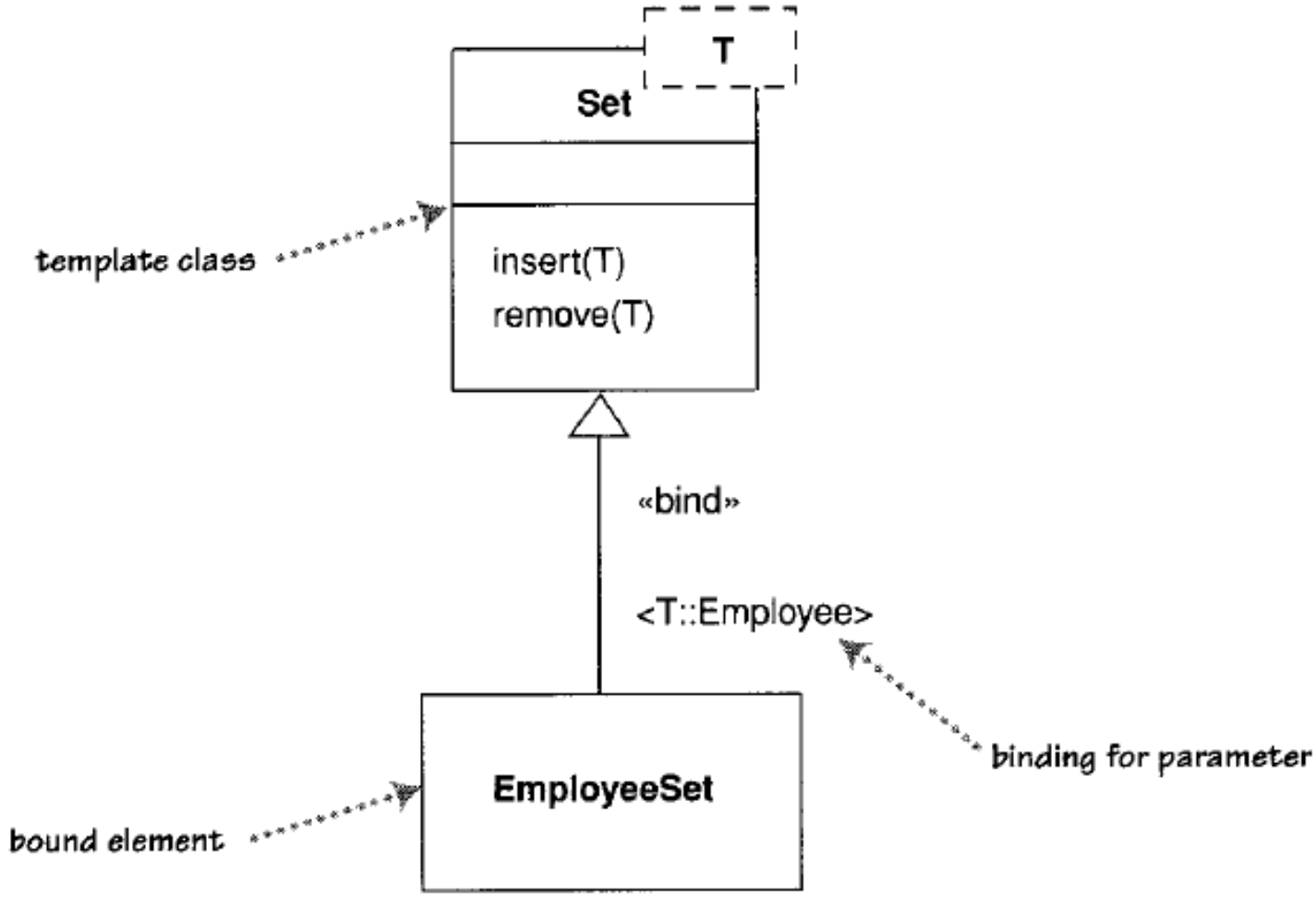
27 Design Patterns

- Administrative details
- Specification using UML
- Design patterns

Administrative Details

- Today's slide are partially based on slides by Dr. Wassyng and on van Vliet (2000)
- A3
  - Part 1 - Solution: Mar 18
  - Part 2 - Code: due 11:59 pm Mar 26
- A4
  - Your own design and specification
  - Model module for game of Freecell
  - Due April 9 at 11:59 pm

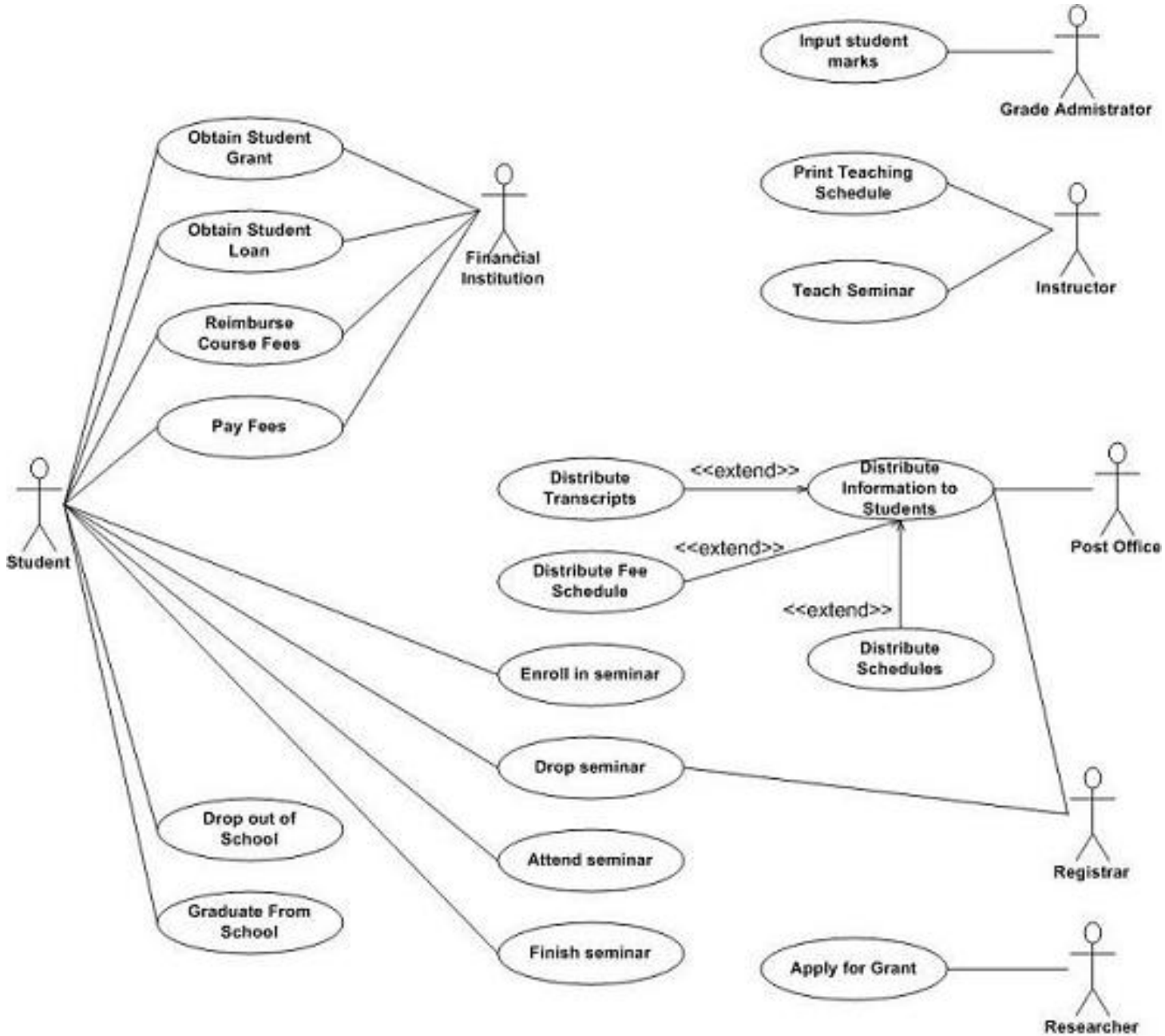
UML Diagram for Generic Classes



UML Class Diagram Template

Use Cases

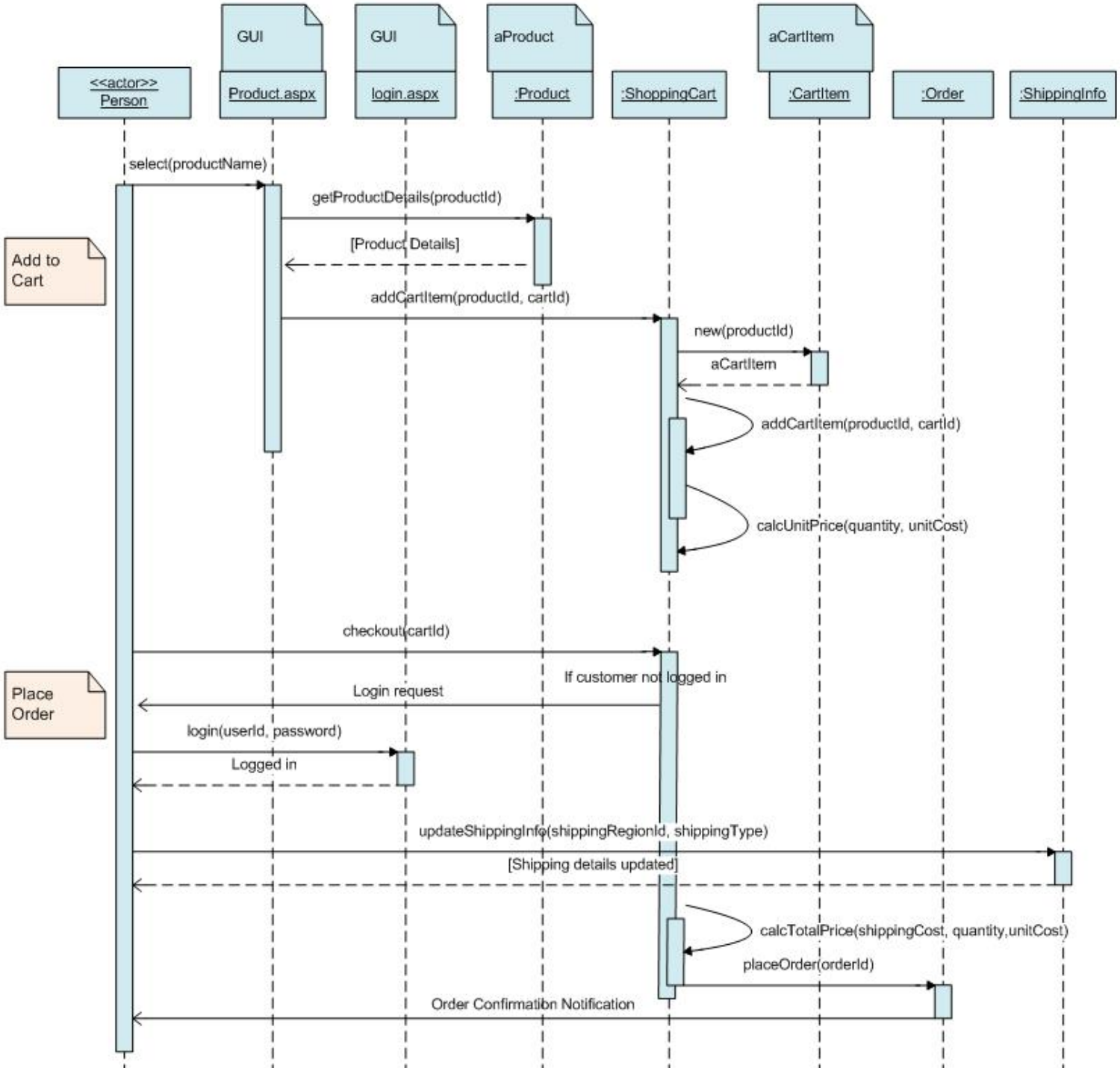
- An overview of the usage requirements for a system
- Made up of:
  - Actors - person, organization, external system
  - Use cases - action to be performed
- Example of University Enterprise Resource Planning (ERP) software (Mosaic)
  - Actors?
  - Use cases?



UML 2 Use Case Diagrams: An Agile Introduction

Use Cases

- Often used for capturing requirements
- From user's (actor's) viewpoint
  - Person
  - Other system
  - Hardware
  - etc. (anything external
- Each circle is a use case
- Lines represent possible interactions
- An actor represents a role, individuals can take on different roles


GUI
GUI
aProduct
aCartItem
<<actor>>
Person
Product.aspx
login.aspx
:Product
:ShoppingCart
:CartItem
:Order
:ShippingInfo
select(productName)
getProductDetails(productId)
[Product Details]
Add to Cart
addCartItem(productId, cartId)
new(productId)
aCartItem
addCartItem(productId, cartId)
calcUnitPrice(quantity, unitCost)
checkout(cartId)
If customer not logged in
Place Order
Login request
login(userId, password)
Logged in
updateShippingInfo(shippingRegionId, shippingType)
[Shipping details updated]
calcTotalPrice(shippingCost, quantity,unitCost)
placeOrder(orderId)
Order Confirmation Notification

Sequence Diagram Question

- Is a sequence diagram an operational or a descriptive specification?
- If objects exchange a message, should there be an association between their classes?

Sequence Diagrams

- Represents a specific use case scenario
- How objects interact by exchanging messages
- Time progresses in the vertical direction
- The vertically oriented boxes show the object's lifeline

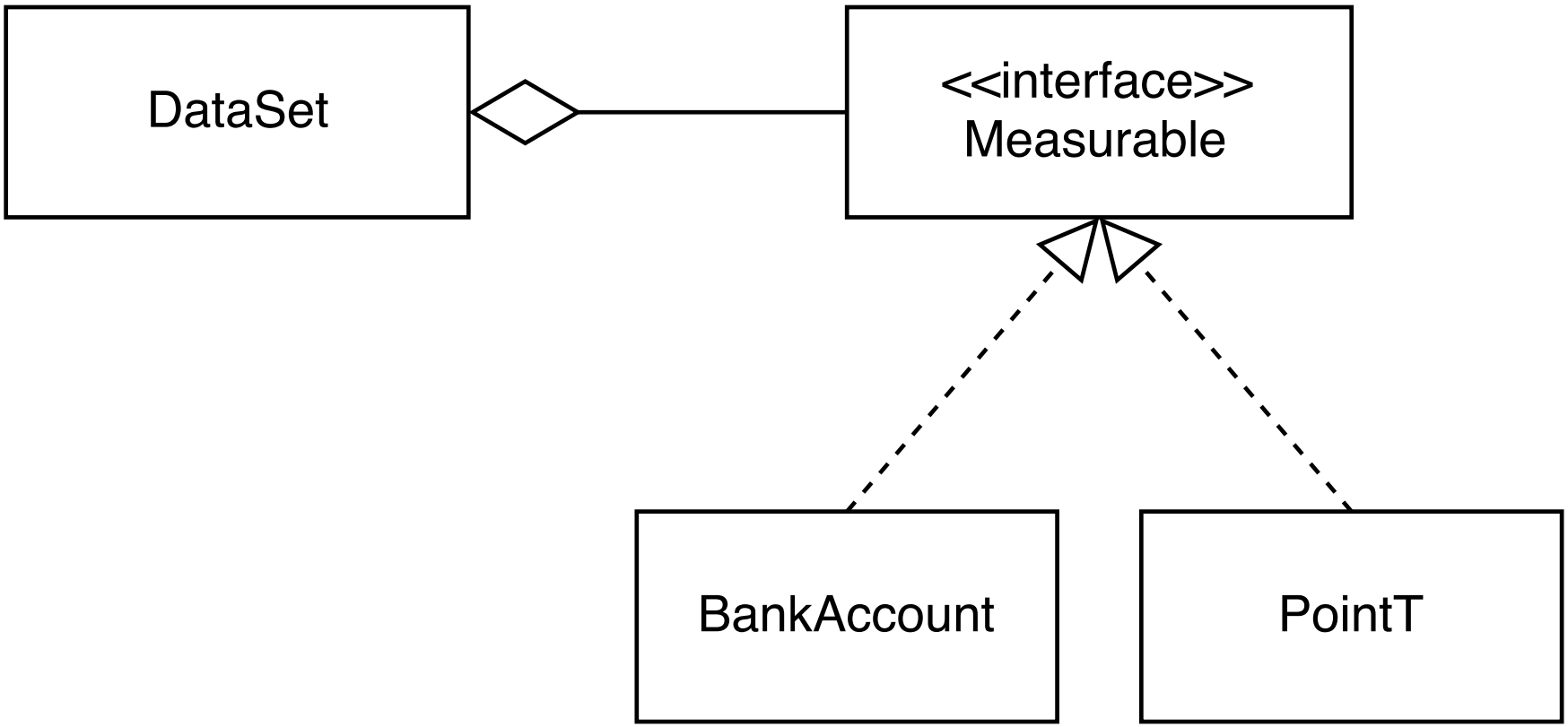
Design Patterns

- Christopher Alexander (1977, buildings/towns):
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way."
- Design reuse (intended for OO)
- Solution for recurring problems
- Transferring knowledge from expert to novice
- A design pattern is a recurring structure of communicating components that solves a general design problem within a particular context
- Design patterns consist of multiple modules, but they do not constitute an entire system architecture

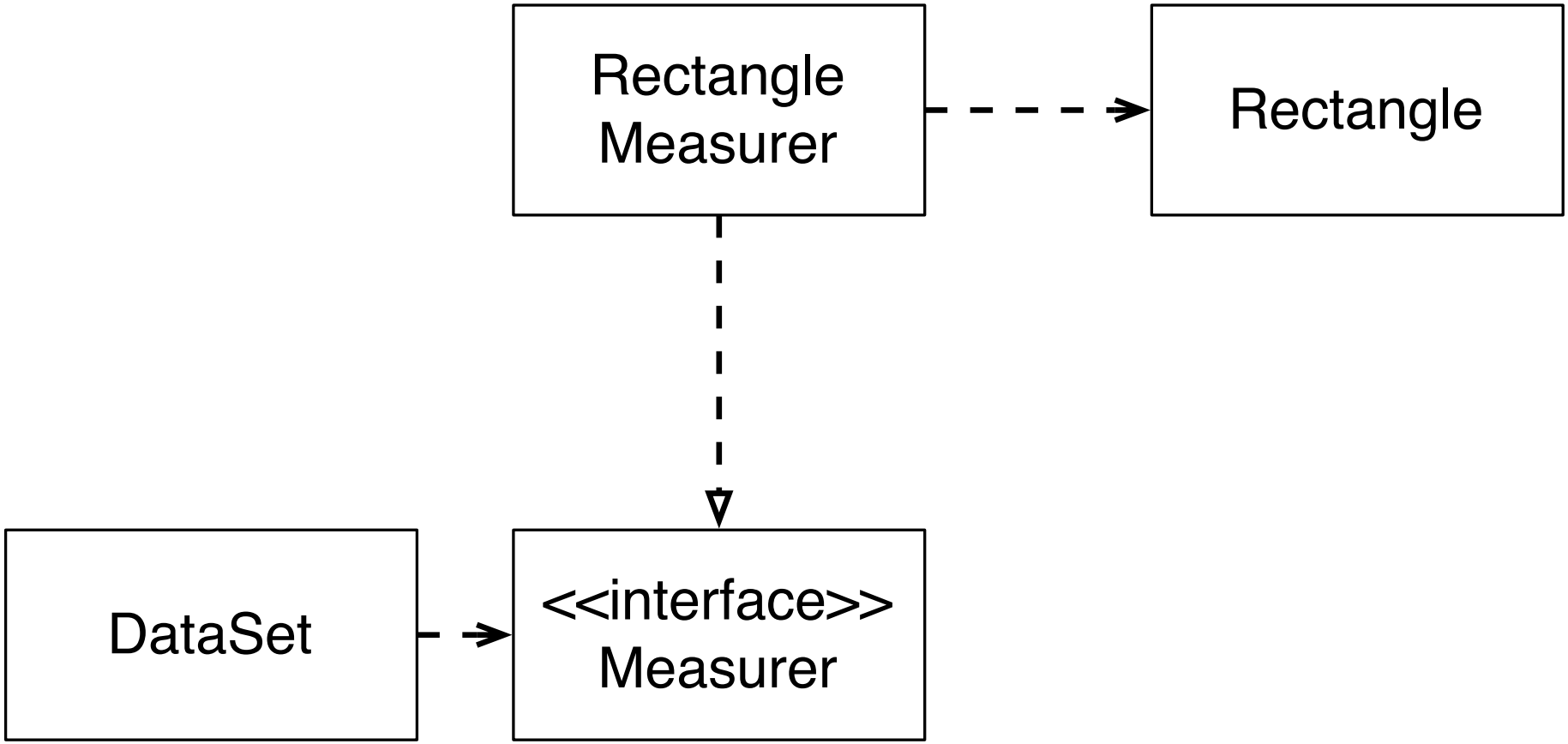
Strategy Design Pattern

- From Source Making web-page
- Define a family of algorithms, encapsulate each one, and make them interchangeable.
- Strategy lets the algorithm vary independently from the clients that use it.
- Capture the abstraction in an interface, bury implementation details in derived classes.
- Where have we used this pattern?

UML Diagram of Measurable Interface



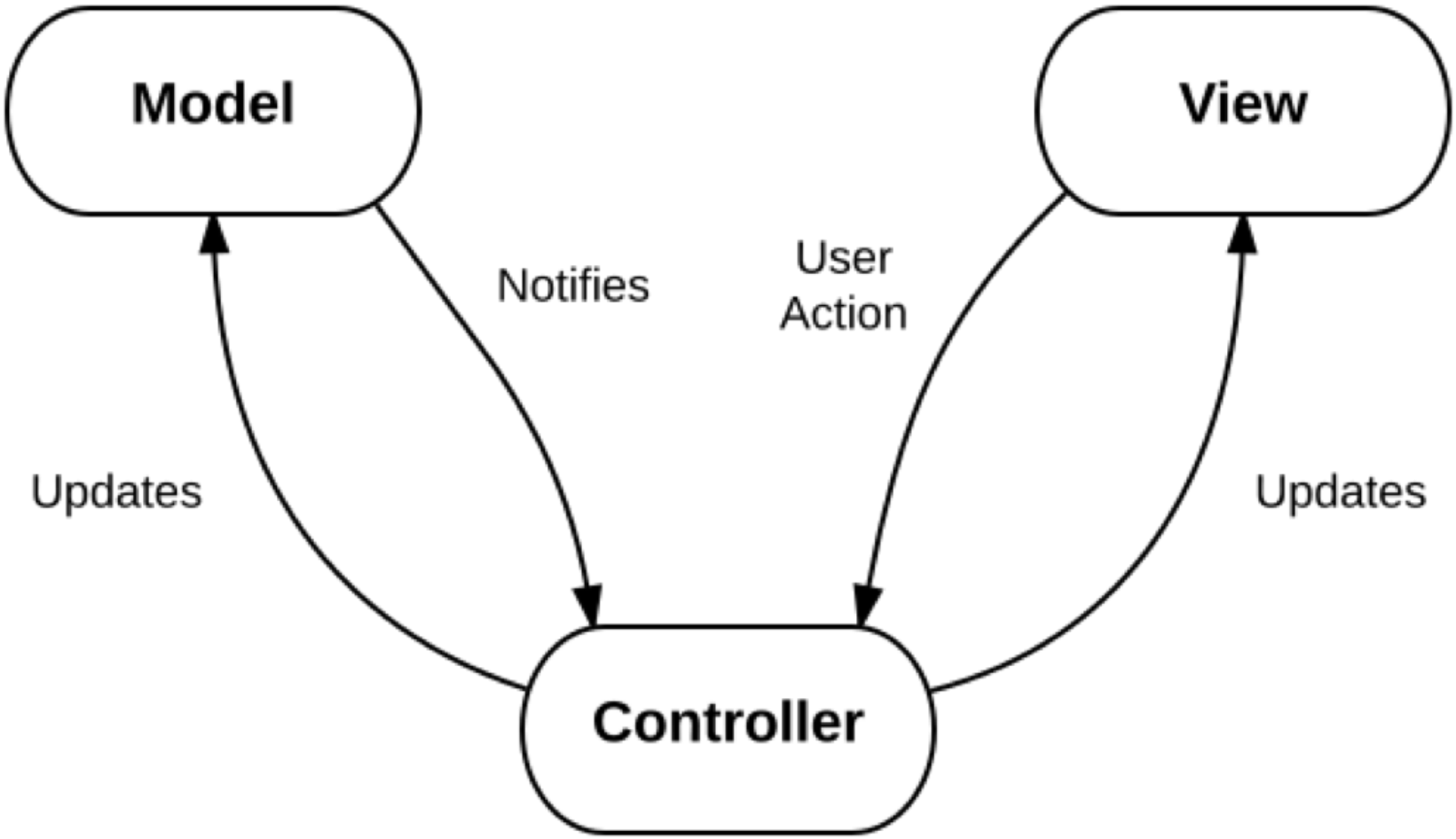
UML Diagram of Measurer Interface



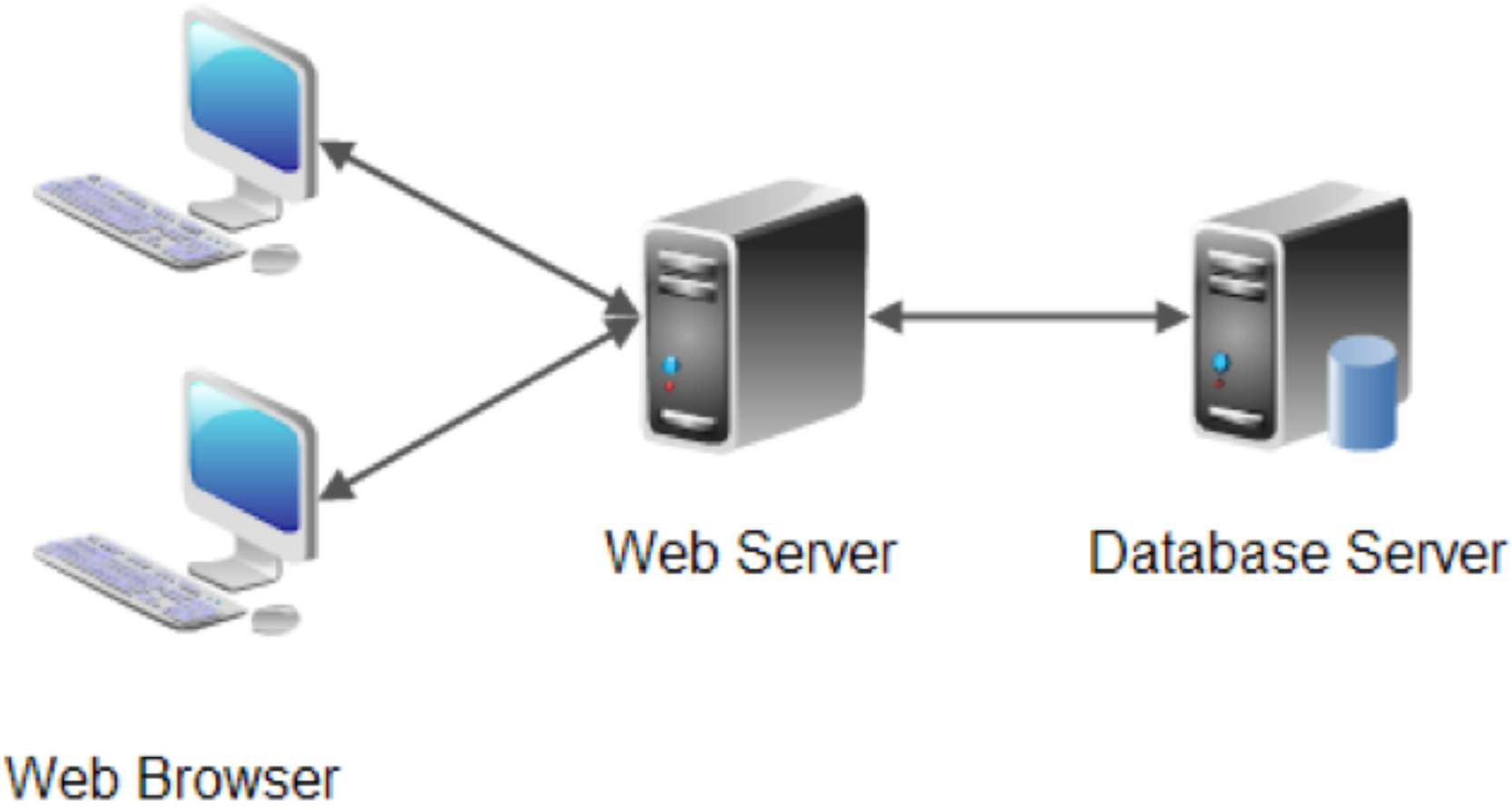
Model View Controller (MVC)

- Separate computational elements from I/O elements
- Three components
  1. Model encapsulates the system's data as well as the operations on the data
  2. View displays the data from the model components, possibly multiple view components
  3. Controller handles input actions
- The controller may or may not depend on the state of the model
- The controller depends on model state when menu items are enabled or disabled depending on the state of the model

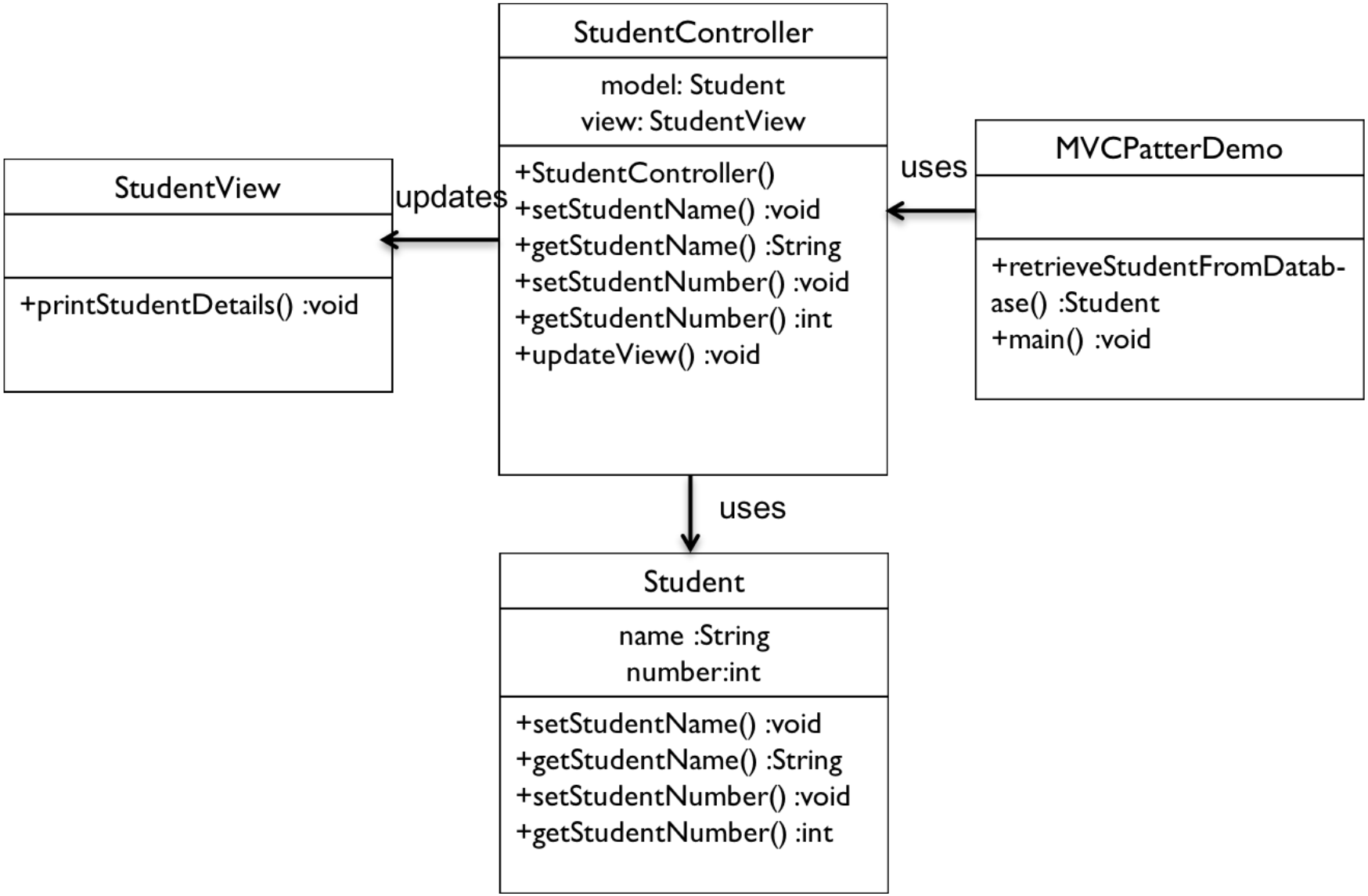
MVC


Model
View
Notifies
User
Action
Updates
Updates
Controller

MVC Web Applications



MVC Example



https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

MVC Critique

- Advantages
  - Simultaneous develoment
  - High cohesion
  - Low coupling
  - Ease of modification
  - Multiple views for a model
- Disadvantages
  - Code navigability
  - Multi-artifact consistency
  - Pronounced learning curve

Wikipedia page

Design Pattern Properties

- A pattern addresses a recurring design problem that arises in specific design situations and presents a solution to it
- A pattern must balance a set of opposing forces
- Patterns document existing, well-proven design experience
- Patterns identify and specify abstractions above the level of single components (modules)
- Patterns provide a common vocabulary and understanding for design principles
- Patterns are a means of documentation
- Patterns support the construction of software with defined properties, including non-functional requirements, such as flexibility and maintainability

Classification of Patterns

- Creational design patterns
  - Abstract factory
  - Object pool
  - Prototype
  - Singleton
- Structural design patterns
  - Adapter
  - Bridge
  - Facade
  - Proxy
- Behavioural design patterns
  - Command
  - Iterator
  - Observer
  - State
  - Strategy

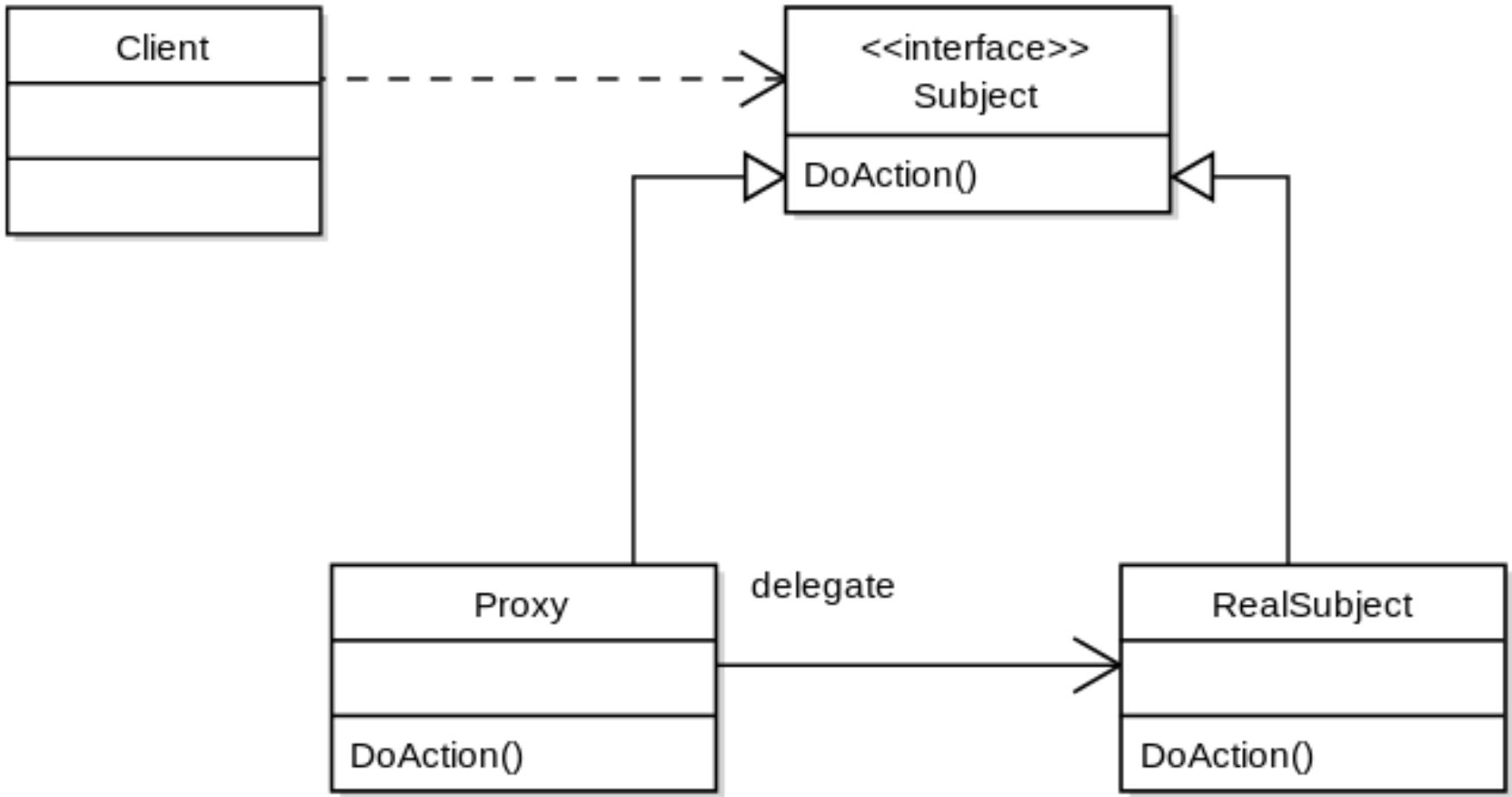
Describing Patterns

- Context: the situation giving rise to a design pattern
- Problem: a recurring problem arising in that situation
- Solution: a proven solution to that problem

The Proxy Pattern (from van Vliet (2000))

- Context: A client needs services from another component. Though direct access is possible, this may not be the best approach
- Problem: We do not want to hard-code access to a component into a client. Sometimes, such direct access is inefficient; in other cases it may be unsafe. This inefficiency or insecurity is to be handled by additional control mechanisms, which should be kept separate from both the client and the component to which it needs access.
- Solution: The client communicates with a representative rather than the component itself. This representative, the proxy, also does and pre- and postprocessing that is needed.
- Code

UML Diagram of Proxy



Command Processor Pattern

- Context: User interfaces which must be flexible or provide functionality that goes beyond the direct handling of user functions. Examples are undo facilities or logging functions
- Problem: We want a well-structured solution for mapping an interface to the internal functionality of a system. All 'extras' which have to do with the way user commands are input, additional commands such as undo and redo, and any non-application-specific processing of user commands, such as logging, should be kept separate from the interface to the internal functionality.

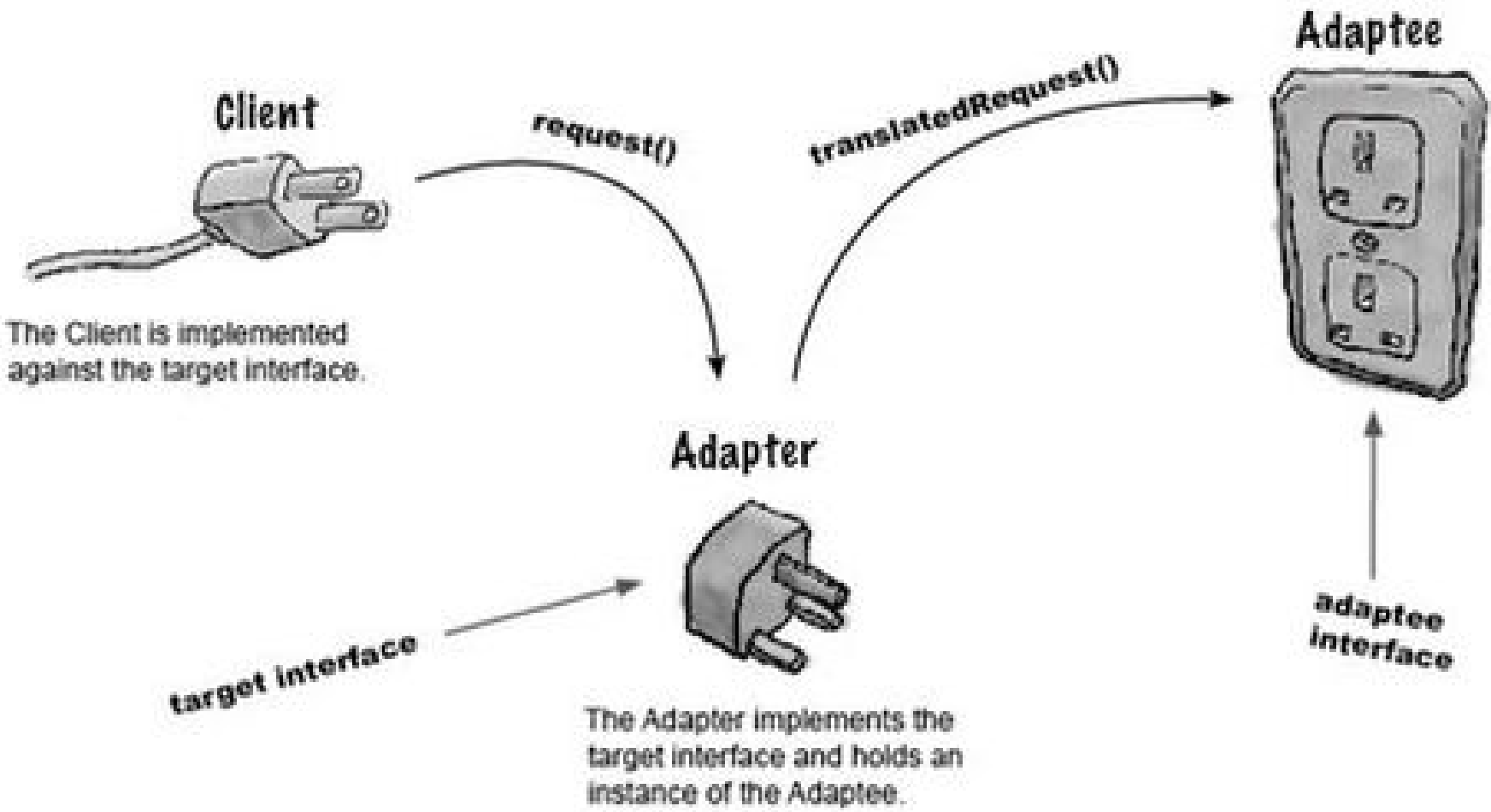
Command Processor Pattern Continued

- Solution: A separate component, the command processor, takes care of all commands. The command processor component schedules the execution of commands, stores them for later undo, logs them for later analysis, and so on. The actual execution of the command is delegated to a supplier component within the application.

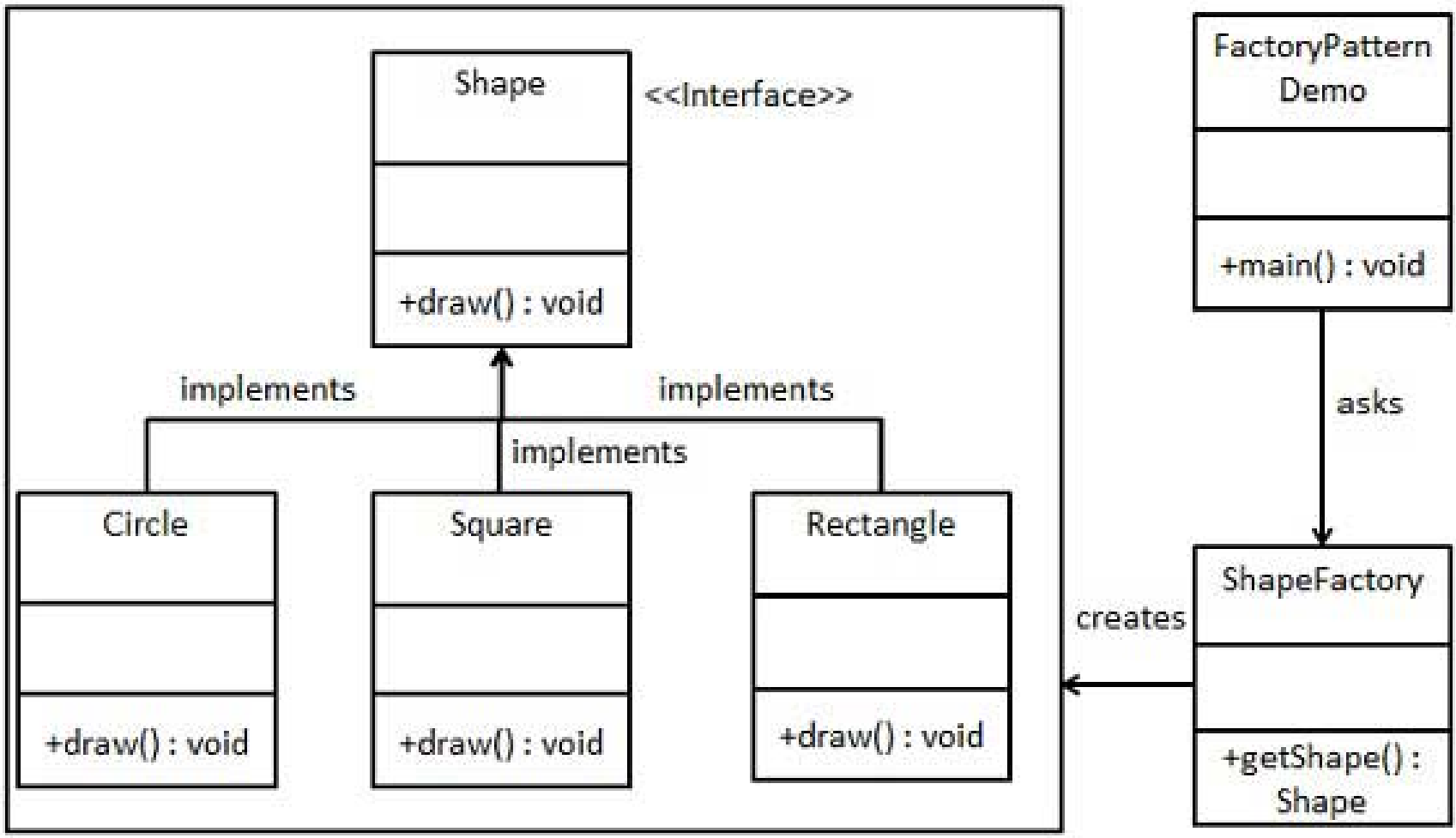
Adapter Design Pattern

When have we used the adapter (or wrapper) design pattern?

Adapter Design Pattern



Factory Pattern



Code

Singleton Pattern