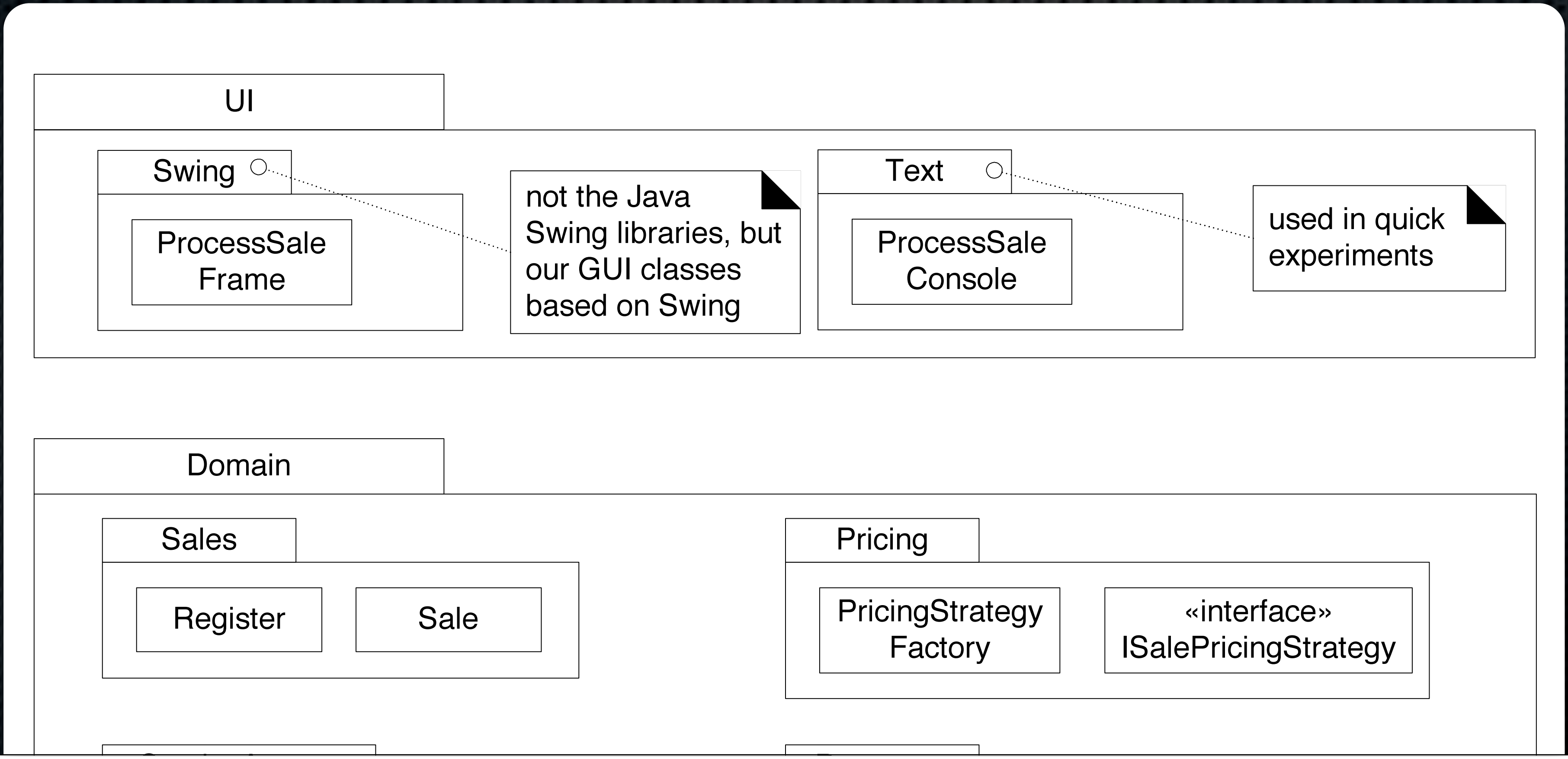
Logical Architecture and Package Design

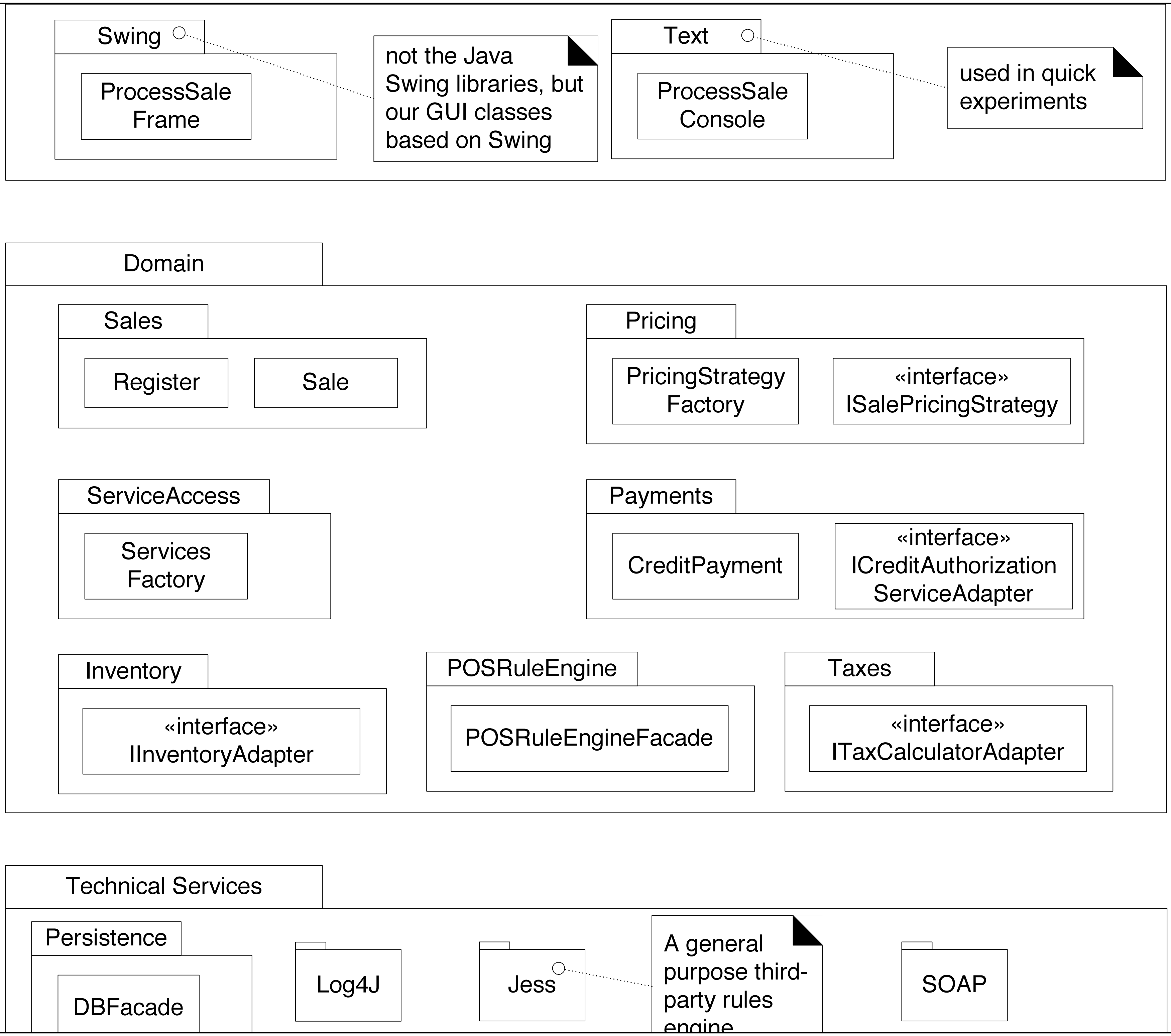
Curt Clifton

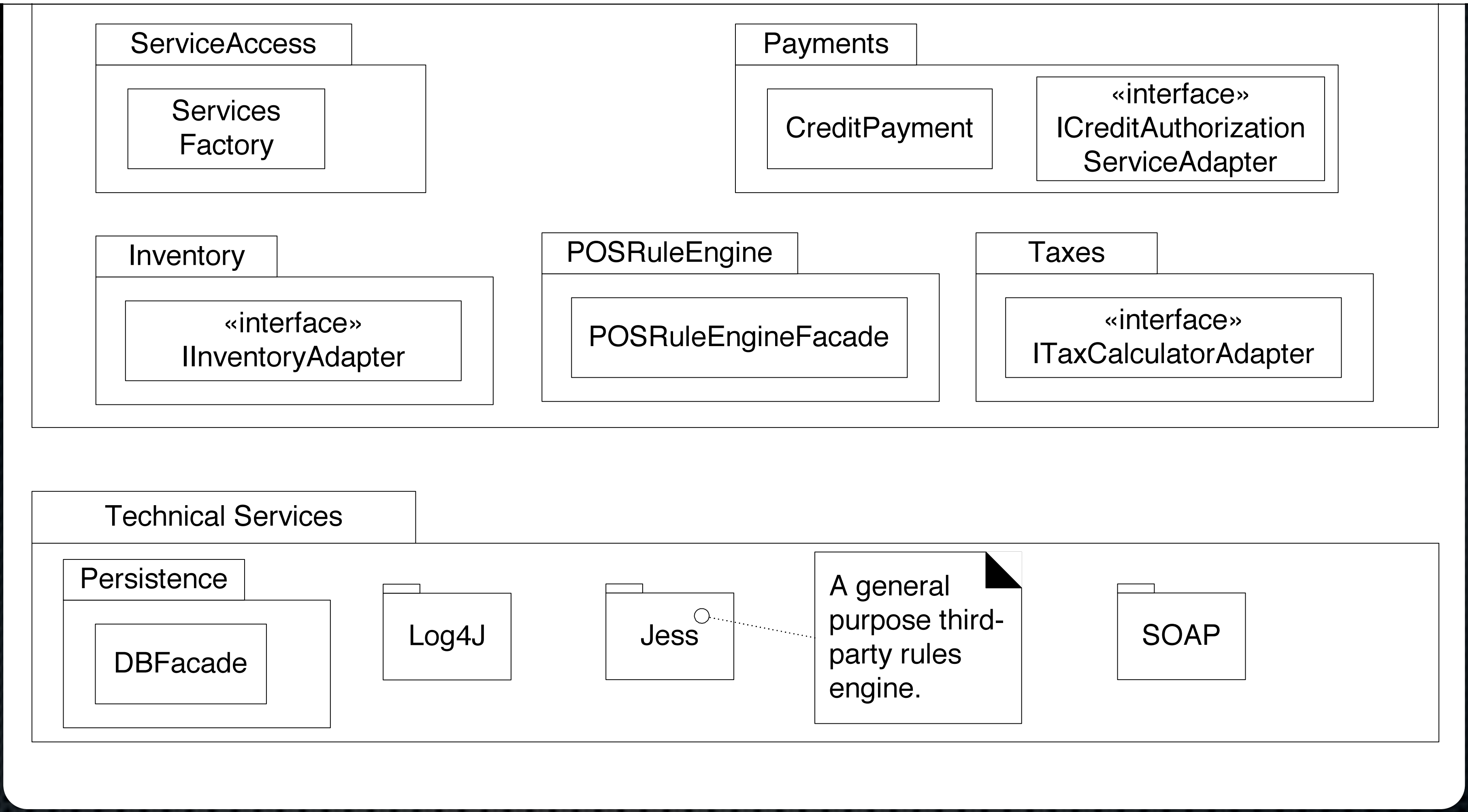
Rose-Hulman Institute of Technology

http://flic.kr/p/2bfN4Q

NextGen POS Logical Architecture


UI
Swing
ProcessSale Frame
not the Java Swing libraries, but our GUI classes based on Swing
Text
ProcessSale Console
used in quick experiments
Domain
Sales
Register
Sale
Pricing
PricingStrategy Factory
«interface» ISalePricingStrategy

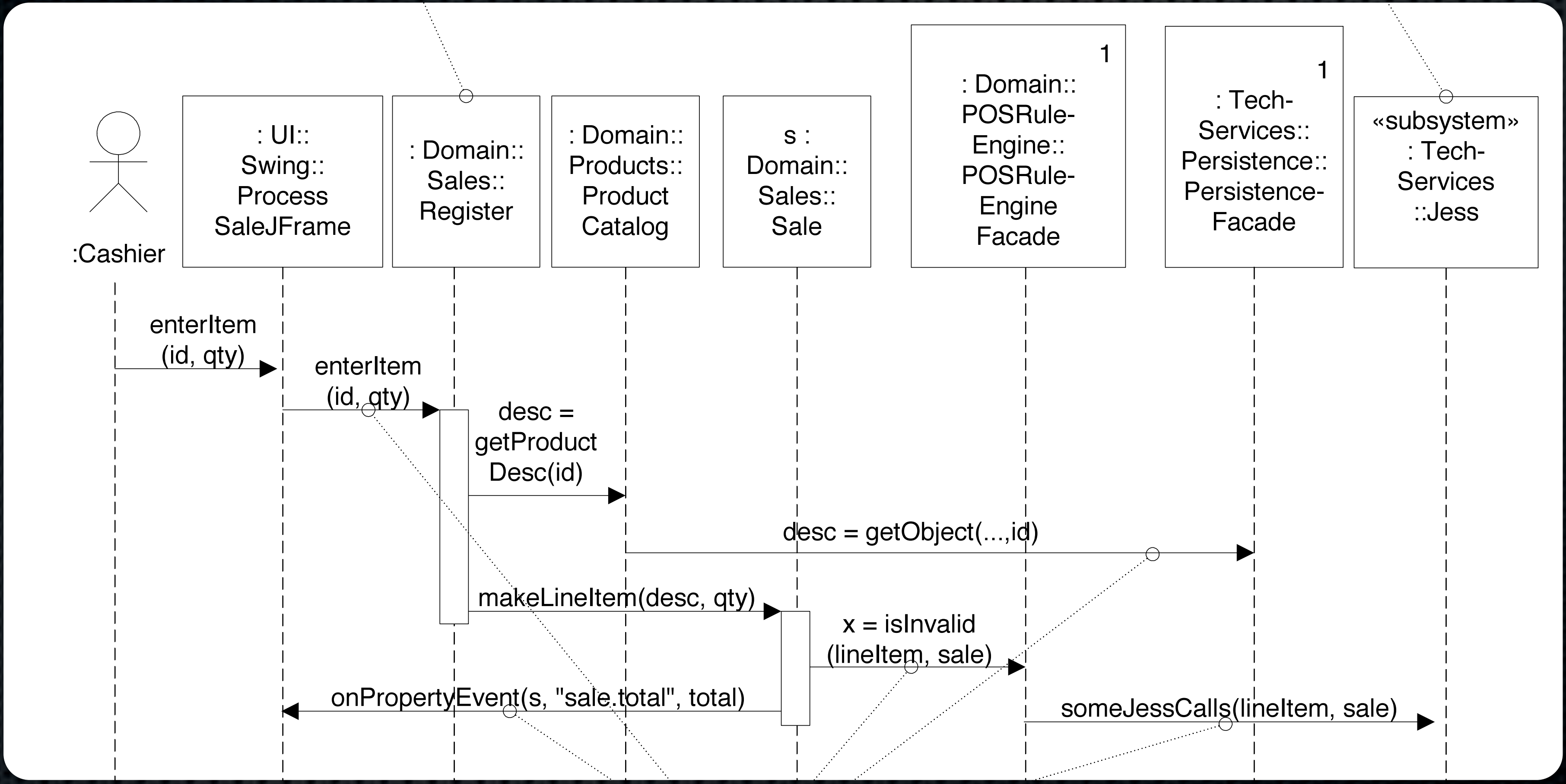

Swing
ProcessSale Frame
not the Java Swing libraries, but our GUI classes based on Swing
Text
ProcessSale Console
used in quick experiments
Domain
Sales
Register
Sale
Pricing
PricingStrategy Factory
«interface» ISalePricingStrategy
ServiceAccess
Services Factory
Payments
CreditPayment
«interface» ICreditAuthorization ServiceAdapter
Inventory
«interface» IInventoryAdapter
POSRuleEngine
POSRuleEngineFacade
Taxes
«interface» ITaxCalculatorAdapter
Technical Services
Persistence
DBFacade
Log4J
Jess
A general purpose third-party rules engine
SOAP



ServiceAccess
Services Factory
Payments
CreditPayment
«interface» ICreditAuthorization ServiceAdapter
Inventory
«interface» IInventoryAdapter
POSRuleEngine
POSRuleEngineFacade
Taxes
«interface» ITaxCalculatorAdapter
Technical Services
Persistence
DBFacade
Log4J
Jess
A general purpose third-party rules engine.
SOAP

Architectural View Diagram

Architecturally Significant Scenarios


:Cashier
: UI:: Swing:: Process SaleJFrame
: Domain:: Sales:: Register
: Domain:: Products:: Product Catalog
s : Domain:: Sales:: Sale
1
: Domain:: POSRule-Engine:: POSRule-Engine Facade
1
: Tech-Services:: Persistence:: Persistence-Facade
«subsystem» : Tech-Services ::Jess
enterItem (id, qty)
enterItem (id, qty)
desc = getProduct Desc(id)
desc = getObject(...,id)
makeLineItem(desc, qty)
x = isInvalid (lineItem, sale)
onPropertyEvent(s, "sale.total", total)
someJessCalls(lineItem, sale)

Q1

Design Decisions at the Architectural Level

- What are the big parts?
  - E.g., Layers
- How are they connected?
  - E.g., Façade, Controller, Observer

Recall: Common Layers

- UI
- Application
- Domain
- Business Infrastructure
- Technical Services
- Foundation

Systems will have many, but not necessarily all, of these

Simple Packages vs. Subsystems

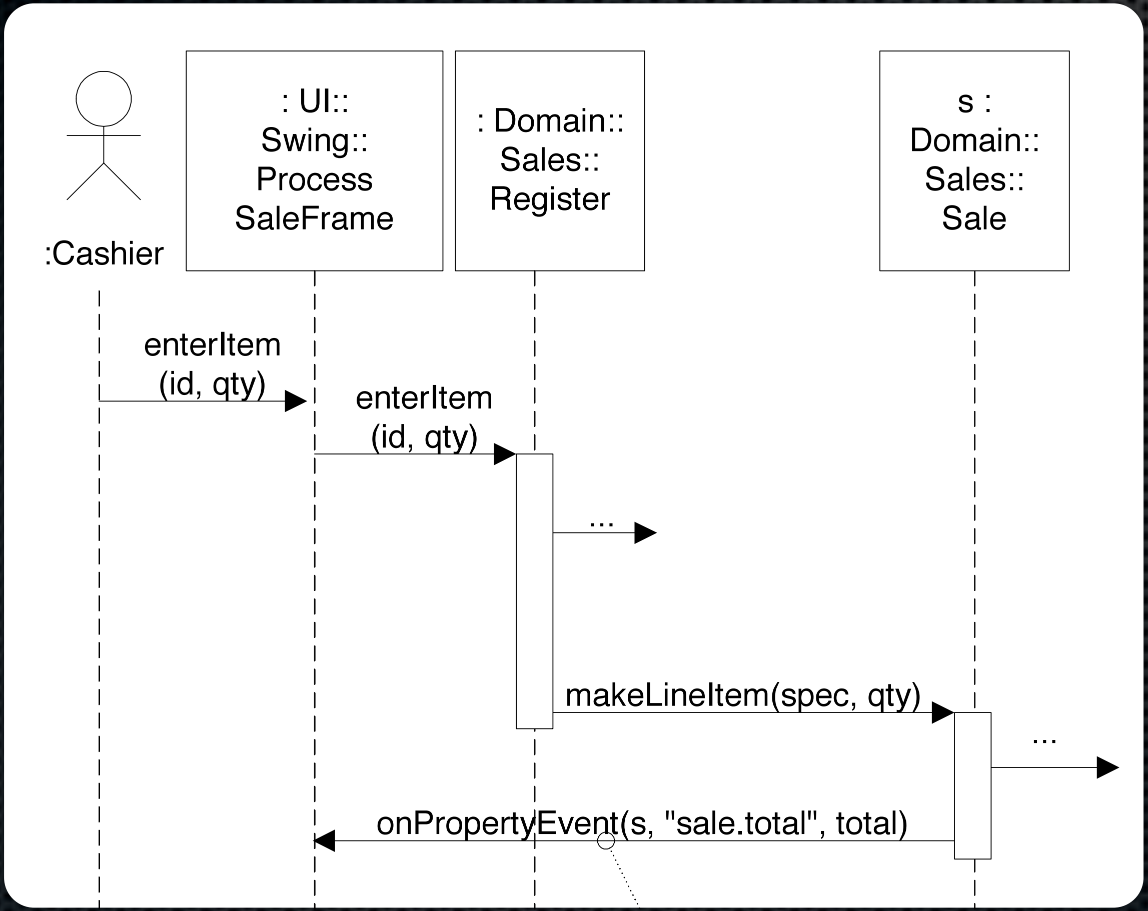
- *Subsystem*: discrete, reusable “engine”
  - Persistence
  - POSRuleEngine
- *Simple package*: just groups classes
  - Pricing
  - Sales

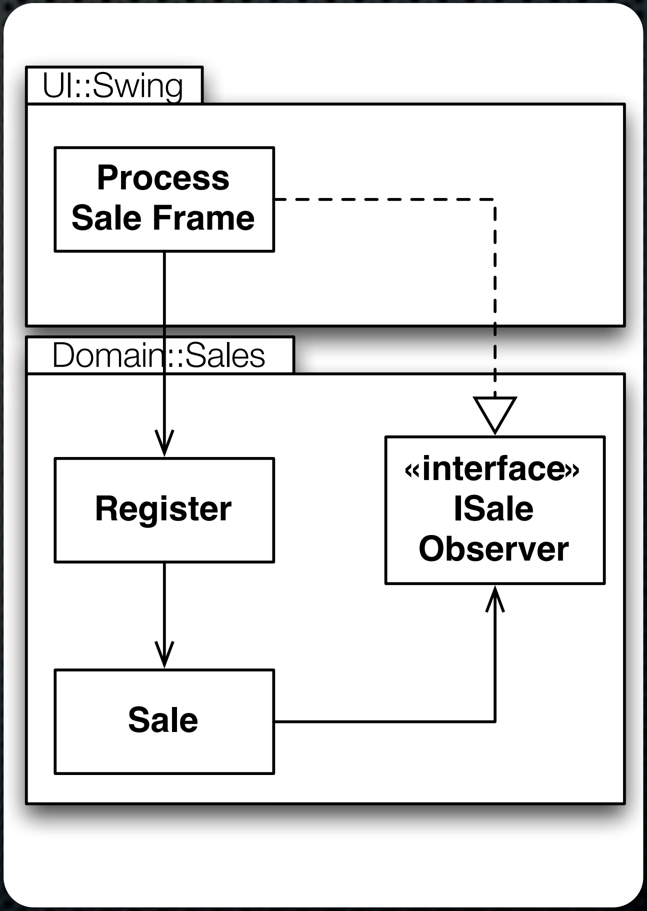
Q2

Subsystems and Façade

- Subsystem packages typically provide a Façade
  - Serves as a single variation point
  - Defines the subsystems services
  - Exposes just a few high-level operations
    - High cohesion
    - Allows different deployment architectures

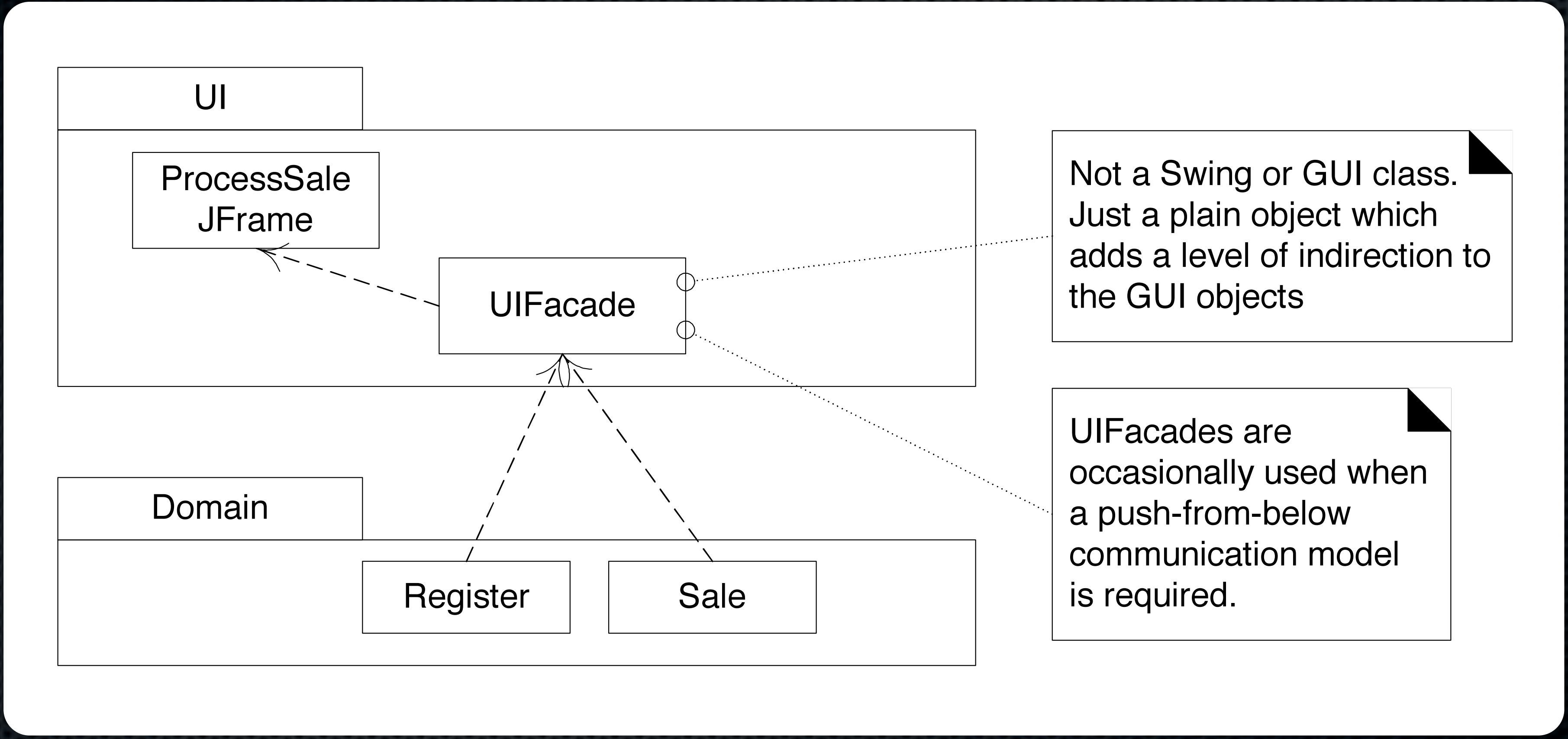
Upward Collaboration with Observer


:Cashier
: UI:: Swing:: Process SaleFrame
: Domain:: Sales:: Register
s : Domain:: Sales:: Sale
enterItem (id, qty)
enterItem (id, qty)
...
makeLineItem(spec, qty)
...
onPropertyEvent(s, "sale.total", total)



Q3

Alternative: Upward Collaboration with UI Façade



Not a Swing or GUI class. Just a plain object which adds a level of indirection to the GUI objects

UIFacades are occasionally used when a push-from-below communication model is required.

For what sort of systems might this be useful?

Application Layer

- Responsibilities:
  - Maintains session state
  - Houses Controllers
  - Enforces order of operations

- Useful when:
  - Multiple UIs
  - Distributed systems with UI and Domain separated
  - Insulating Domain from session state
  - Strict workflow

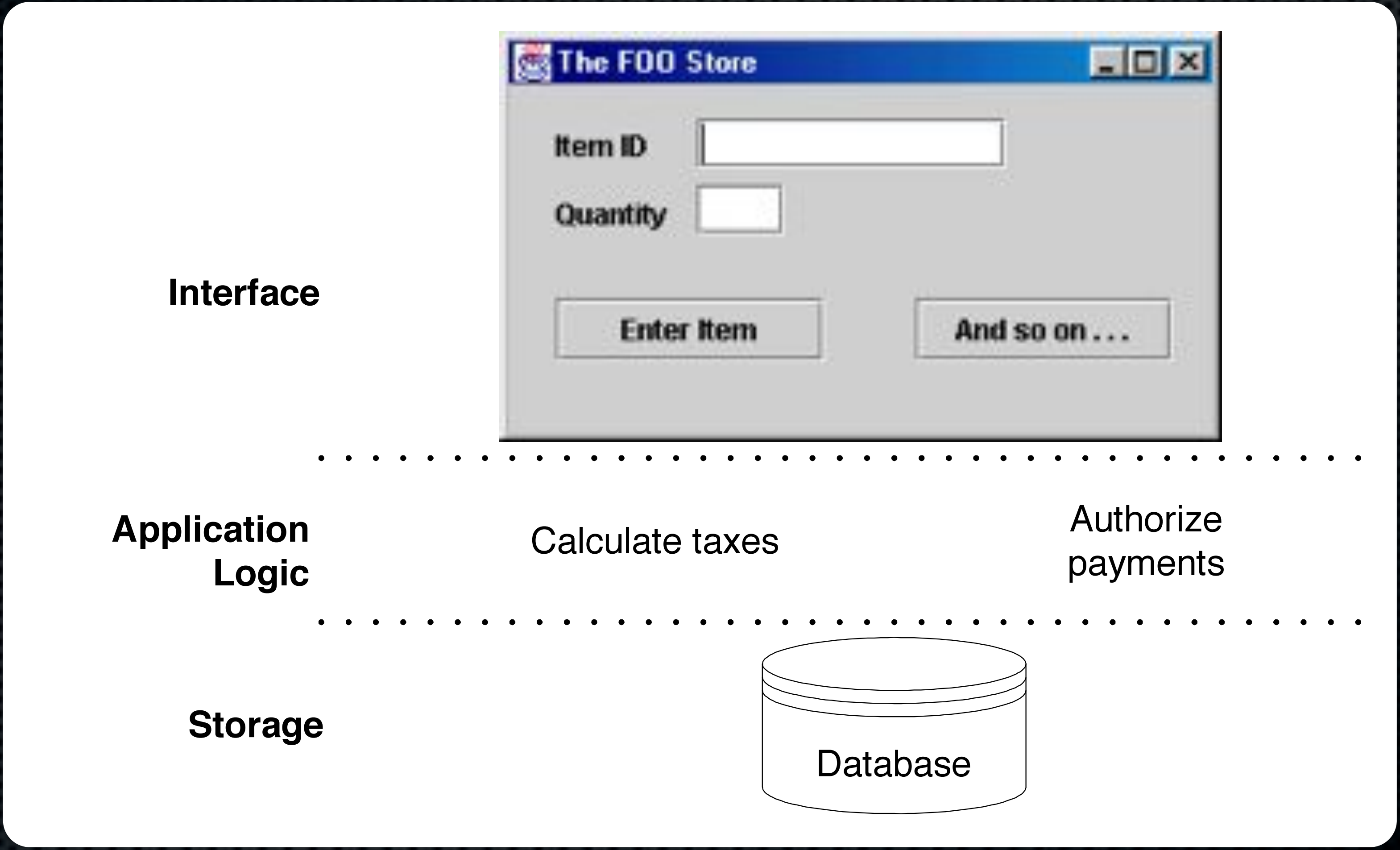
Typical Coupling Between Layers

- From higher layers to Technical Services and Foundation
- From Domain to Business Infrastructure
- From UI to Application and Application to Domain
- In desktop apps: UI uses Domain objects directly
  - E.g., Sales, Payment
- Distributed apps: UI gets data representation objects
  - E.g., SalesData, PaymentData

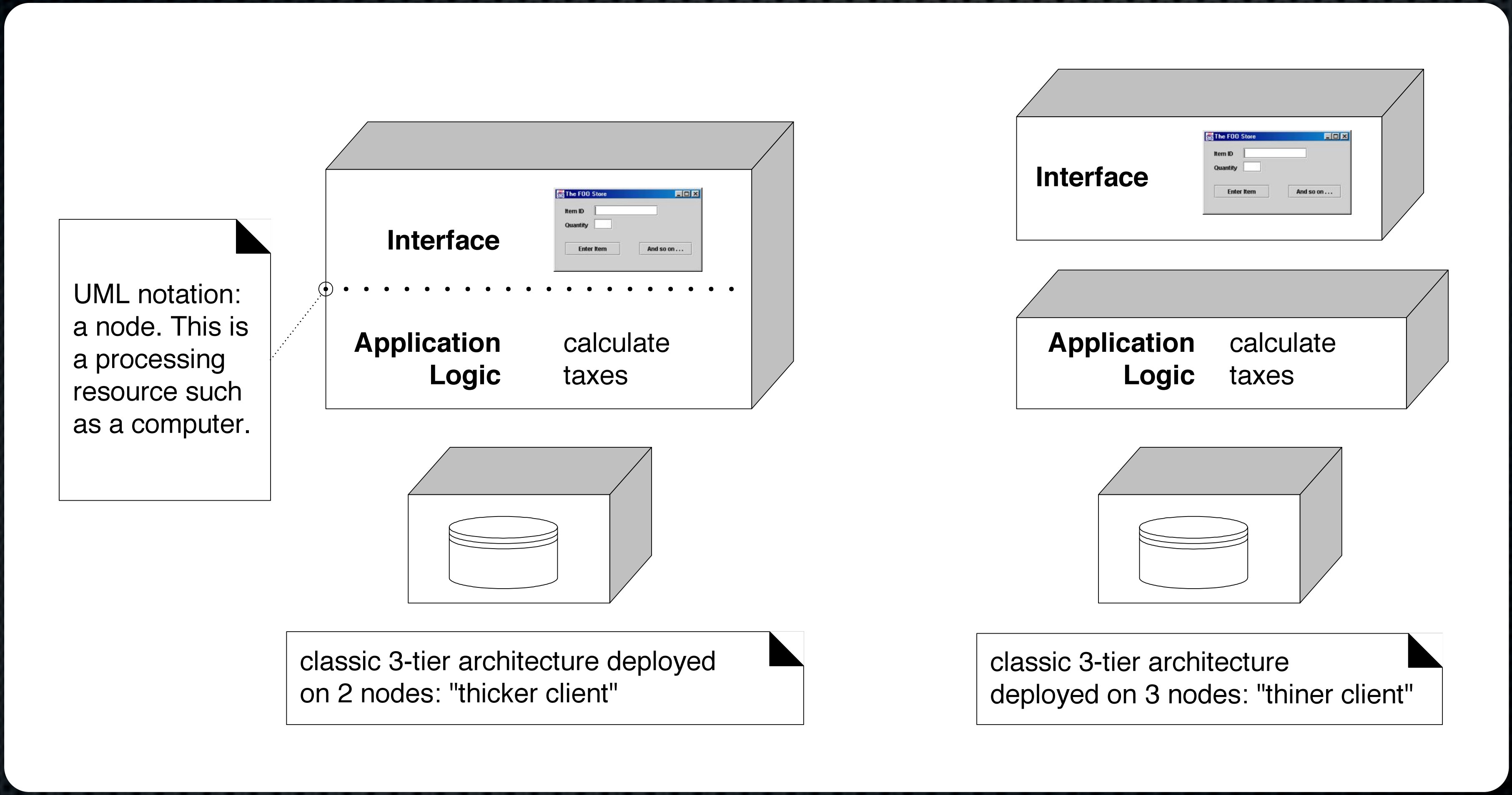
Liabilities with Layers

- Performance
  - E.g., game applications that need to directly communicate with graphics cards
- Poor architectural fit sometimes
  - Batch processing (use “Pipes and Filters”)
  - Expert systems (use “Blackboard”)

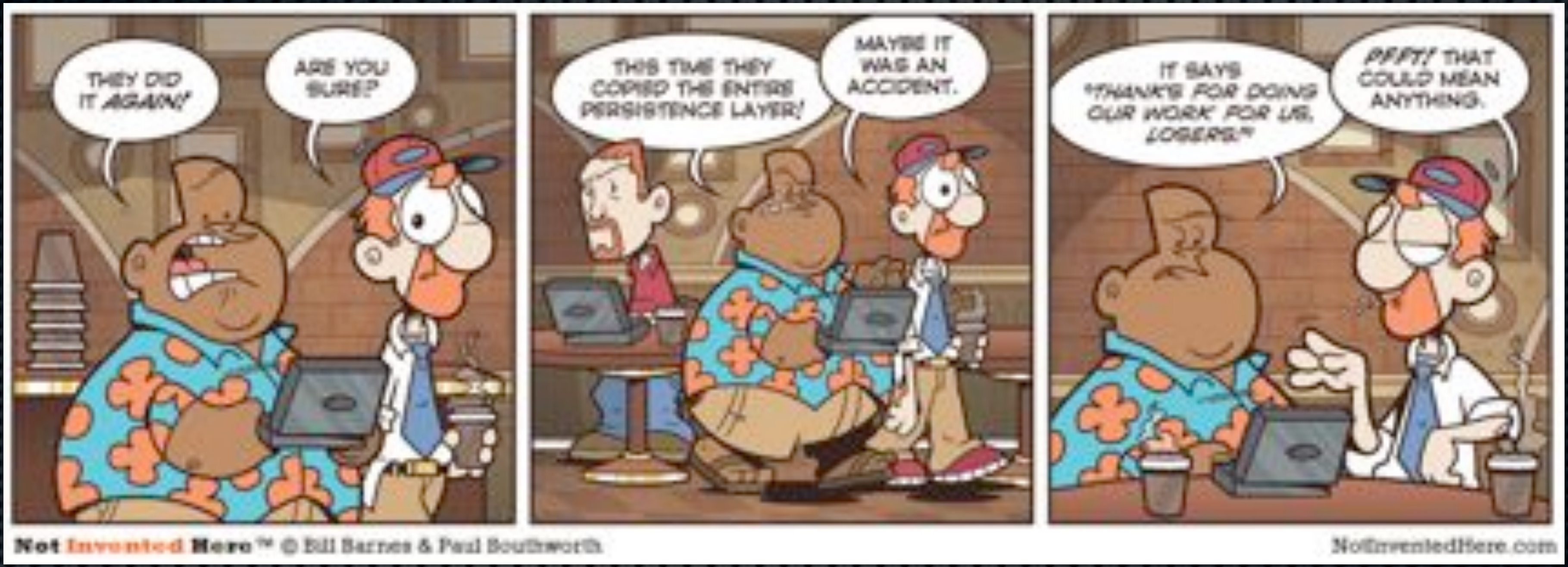
Info. Systems: Classic Three-Tier Architecture



Info. Systems: Classic Three-Tier Architecture


UML notation: a node. This is a processing resource such as a computer.
Interface
The FOO Store
Item ID
Quantity
Enter Item
And so on . . .
Application Logic
calculate taxes
classic 3-tier architecture deployed on 2 nodes: "thicker client"
Interface
The FOO Store
Item ID
Quantity
Enter Item
And so on . . .
Application Logic
calculate taxes
classic 3-tier architecture deployed on 3 nodes: "thiner client"

Cartoon of the Day


THEY DID IT AGAIN!
ARE YOU SURE?
THIS TIME THEY COPIED THE ENTIRE PERSISTENCE LAYER!
MAYBE IT WAS AN ACCIDENT.
IT SAYS "THANKS FOR DOING OUR WORK FOR US, LOSERS."
PFFT! THAT COULD MEAN ANYTHING.
Not Invented Here™ © Bill Barnes & Paul Southworth
NotInventedHere.com

Used by permission. http://notinventedhe.re/on/2009-12-21

Physical Package Design

Multiple logical packages might be developed together physically

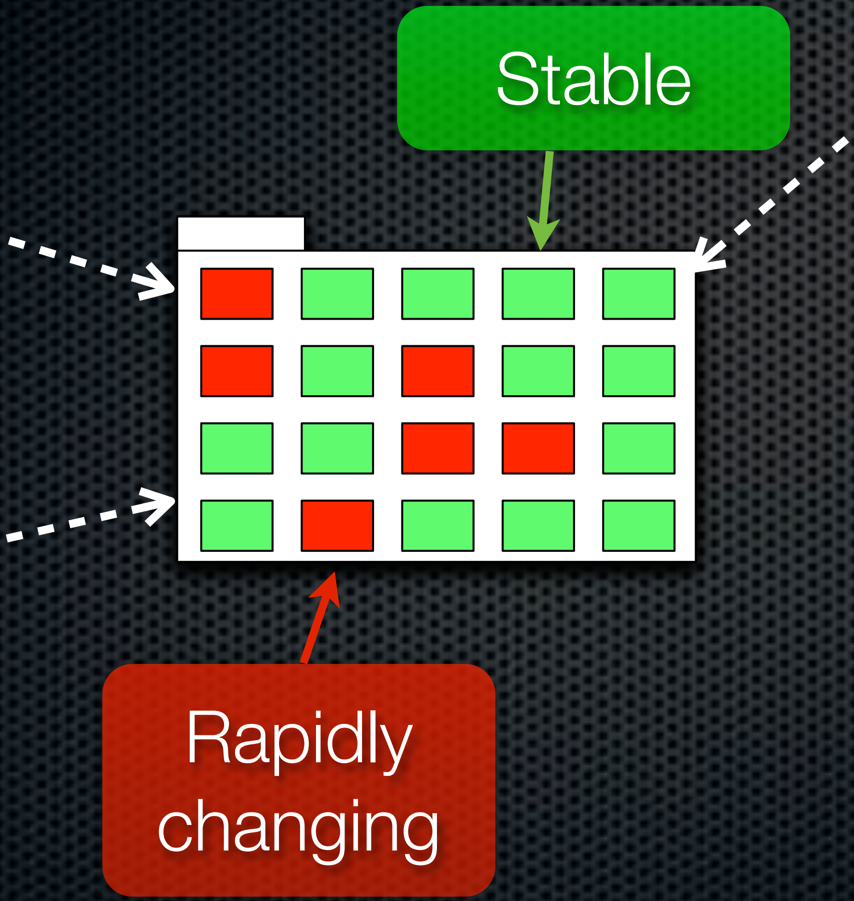
- Goal: define physical packages so they can be:
  - Developed independently
  - Deployed independently
- Packages should depend on other packages that are more stable than themselves
  - Avoids *version thrashing*

Package Organization Guidelines

- Package functionally cohesive slices
  - Keep strong coupling within the package
  - Achieve weak coupling between packages
- Package a family of interfaces
  - Factor out independent types

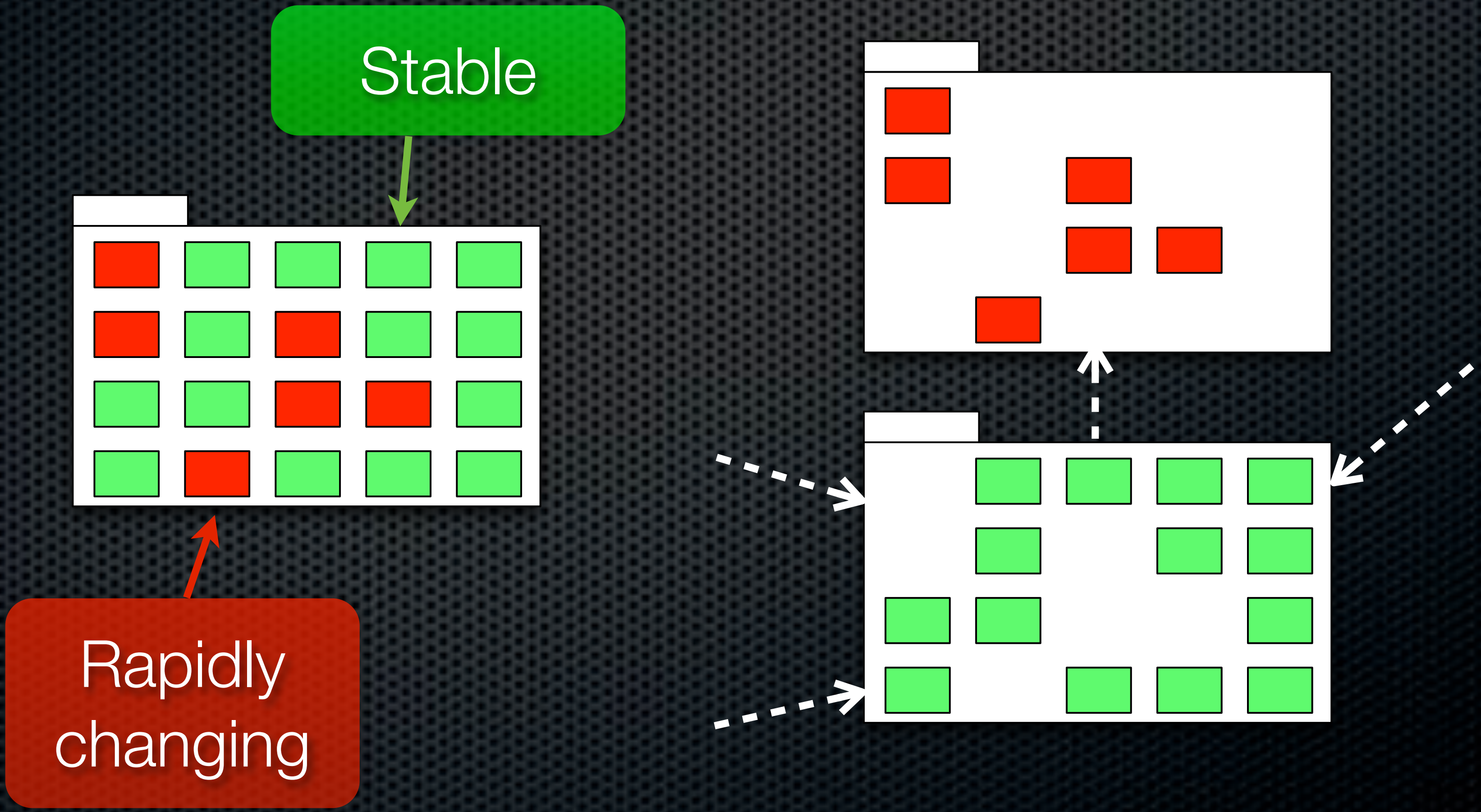
Package Organization Guidelines

- Package by clusters of unstable classes



Package Organization Guidelines

- Package by clusters of unstable classes



Package Organization Guidelines

- Make the most depended-on packages the most stable
- Can increase stability by:
  - Using only or mostly interfaces and abstract classes
  - Not depending on other packages
  - Encapsulating dependencies (e.g., with Façade)
  - Heavy testing before first release
  - Fiat

Iron-fisted rule, not crappy cars

Q5

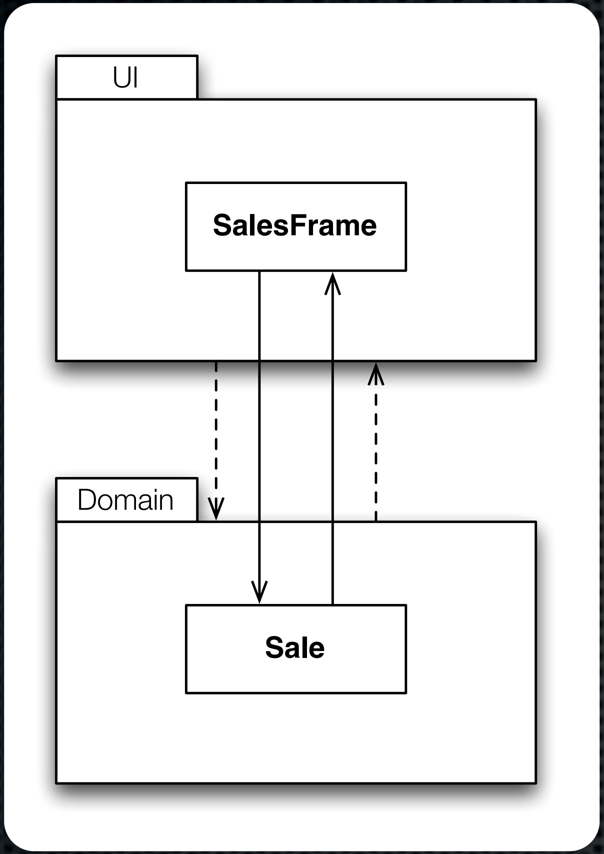
Package Organization Guidelines

- Use factories to reduce dependencies on concrete packages
    - E.g., instead of exposing all the subtypes, expose an abstract superclass and a factory

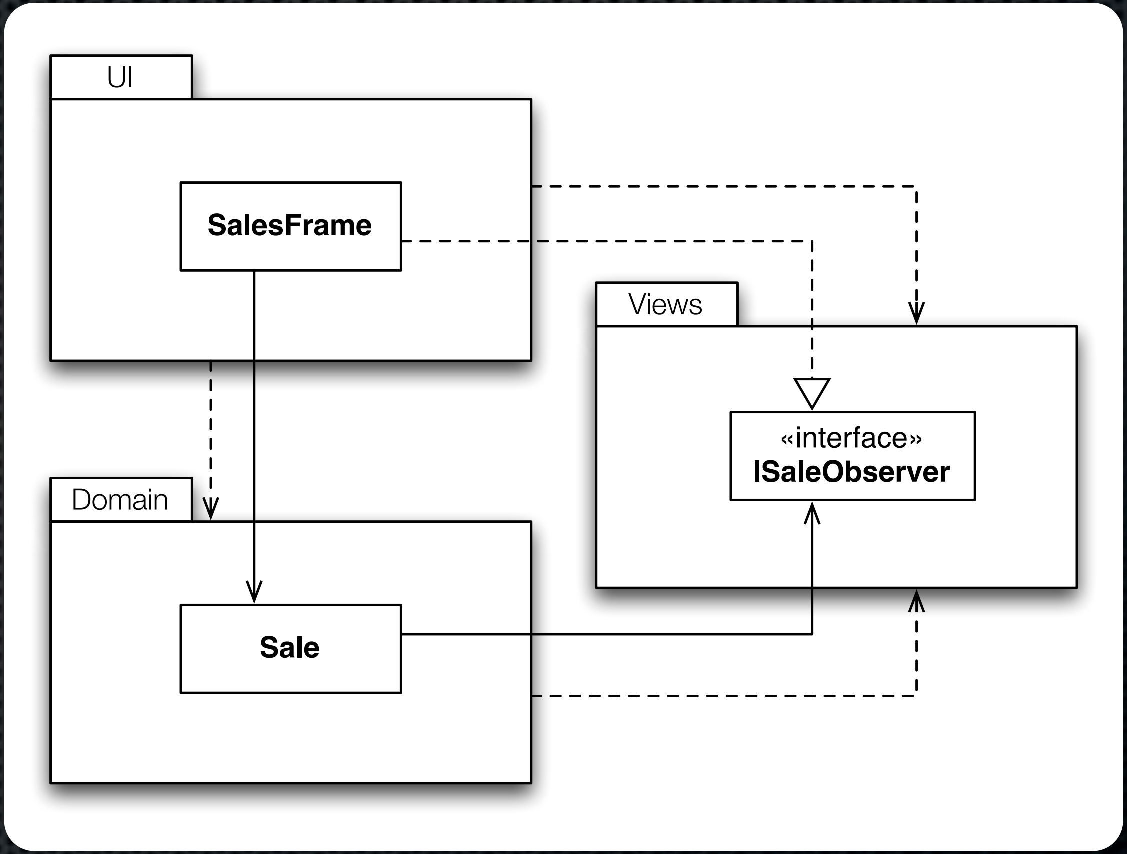
Package Organization Guidelines

- No cycles between packages
  - Cycles often force packages to be developed and released together
- Can use interfaces to break cycles
  - Example…

Breaking Dependency Cycles Between Packages



Cyclic Coupling



Cycle Removed, yay!

Q6

Design Studio: Personal Fitness Tracker

Team describes problem and perhaps current solution (if any)	~5 min.
Class thinks about questions, alternative approaches. **Q7**	~3 min.
On-board design	~12 min.