



# Effective Unit-Testing of In-Container Components

---

Dan Bergh Johnsson  
Omegapoint AB, Sweden





# Intro

---

- Unit tests are like flossing
- Why not practiced more?
- Real life obstacles
- Can be done – try harder
  - Powerful frameworks out there
- Or – try smarter
  - Point out alternative way



# Code examples

---

- <http://www.omegapoint.se/public/css2005>



# Agenda

---

- Mindset
- Unit Tests IRL – Is that a container I see?
- Principles through examples
- Principles applied



# Mind Set: Object Orientation

---

- Keep related things together
- Keep unrelated things apart
- One idea in one place (program unit)
  
- When finding discrepancy: refactor



# Mind Set: Unit Tests

---

- Test that the code behaves as the *programmer* thinks it does
- *aka programmer test* (esp. in XP-circles)
- Tests a unit, *i.e.* one idea (cfr. OO)



# Definition of Unit Test

---

- Characteristics
  - Automated
  - Self checking
  - Fast running
  - Not hitting DB
- Not good for definition
  - Misses some point



# Definition of Refactoring

---

- Fowler “Refactoring”
  - Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
  - Refactoring (verb): to restructure software by applying a series of refactorings without changing its observable behavior.
- Getting closer



# Definition of Unit Test (danj)

---

- Test of the observable and intentional behaviour of one unit of software
  - Behaviour – not code *per se*
  - Observable – not implementation details
  - Intentional (by the programmer) – not what happened-to-be
  - One Unit – not several ideas
- Apology



# Test Behaviour – Not Code

---

- Example `java.util.Set`
- Test of Code (method add)
  - Create, add, ?
- Test of Behaviour (member or not)
  - Create, add, exists
  - Create, add, add, exists
  - Create, add, remove, exists
  - Create, add, add, remove, exists



# Other Tests

---

- Function tests
- Integration tests
- Performance tests (in general NFR-tests)
- (Regression tests)
  
- Most tests that requirements are met
  - Declare what is wanted



# Unit Tests

---

- Money
- MoneyTest extends TestCase
- So simple, and beautiful ...
- Problem: world not that simple



# In-container Components

---

- IRL functions embedded in components
  - EJB
  - Struts Action
- Components live in containers
  - Cannot live without
- Shopping cart example
  - Session EJB
- Real Life ShoppingCart
  - Probably persistent
- How do we test our components?



# Testing of Components

---

- Direct approach
  - Test the code “as it is”
- More advanced environment
- More advanced testing
- More advanced methods/tools
- Example
  - Deploy in real container
  - Drive test suite from outside



# Problem

---

- Slow execution (db-setup)
- Slow roundtrip (rebuild, redeploy, restart...)
- Awkward rigging
  - Other components (mock, real)
  - Database – maintain testdata – migrate



# Overkill Alert

---

- *What* do we test?
  - Arithmetic operations
  - Conditionals
  - Iteration/recursion
- Kind of overkill

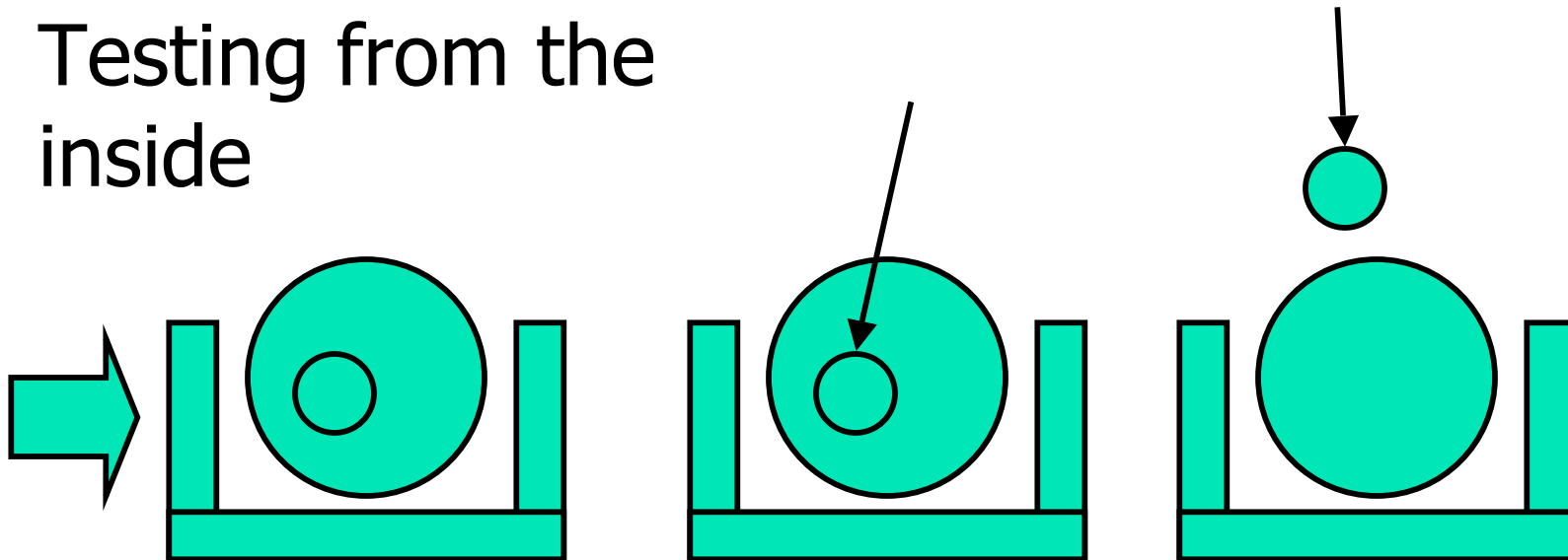
# Rethinking Unit Tests of Components

---

- Do not test code
- Test an idea
- Test just the idea
  
- Sounds reasonable ...
- ... but how?

# Two Ways of Testing

- Testing from the outside
- Testing from the inside





# Some Principles – through Examples

---

- Method (recursion case)
- Endpoint (base case)



# Scenario: Business Logic in EJB

---

- Business logic often hard to get right
  - Fowler PoEAA: “business illogic”
  - Need for high coverage (quality – not quantity)
  
- Business tier complications
  - “interlocked” logic
    - Logic depend on each other
  - Other services
    - Security
    - Transactions
    - Connection (*e.g.* DB)
  
- Functionality in EJB hard to test
  - Rod Johnson “J2EE Development without EJB”



# Example: Original Design

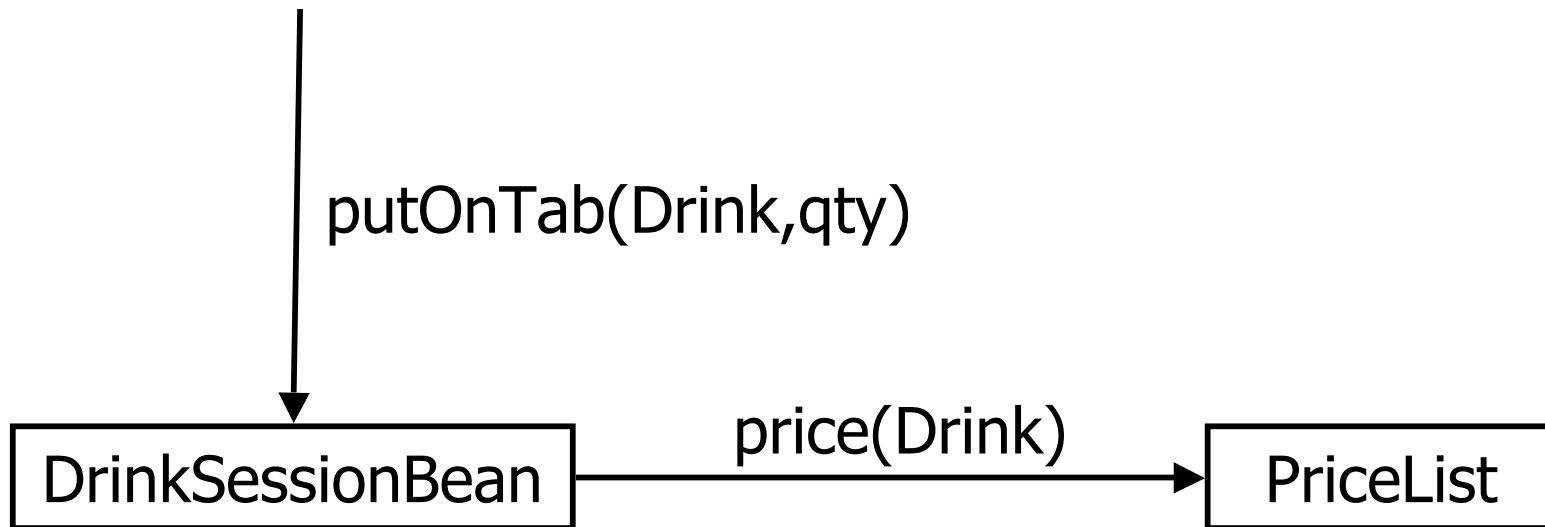
---

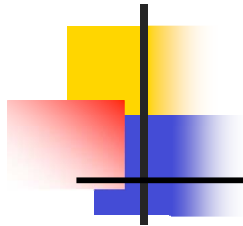
- SFSB DrinkingSession
  - create(credit-limit)
  - openTab()
  - putOnTab(Drink, Qty)
  - getTab()
  - closeTab(CreditCard)
    - Not possible to put more on
- Price calculation
- Credit-limit on each tab
- (Cover drink-minimum)
- One tab at a time
- Can only put drinks on open tab



# Design

---





# CODE!

---

## DrinkSessionBean

- Remember/enforce credit limit
  - int creditlimit
- Keep track of tab or not
  - boolean hasTab
- Keep track of sum
  - int tabSum

DrinkSessionBean
int creditlimit boolean hasTab int tabSum

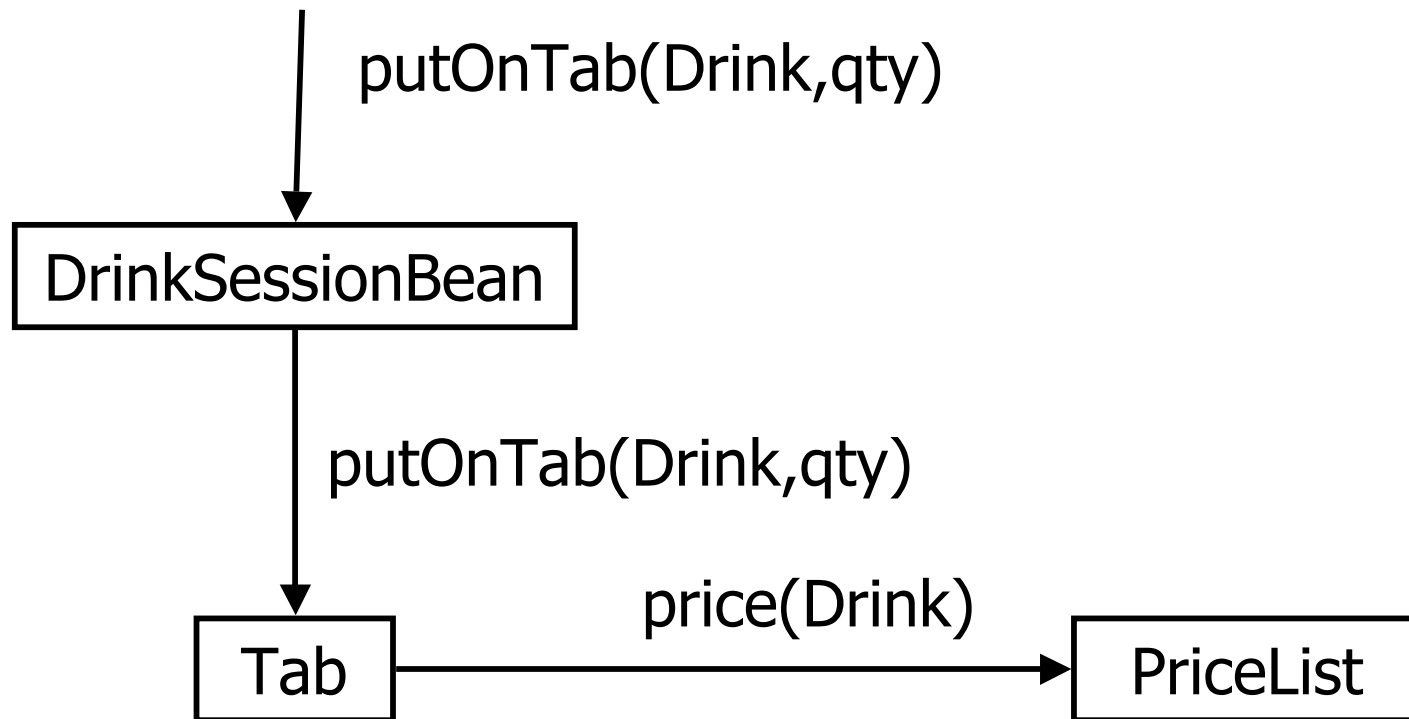


# Problem and Solution

---

- Tab logic kind of complex – need tests
- Complexity “in way”
  - EJB environment
  - JNDI lookup
  - Access to PriceList
  
- OO rule
  - each idea isolated
  - keep separate things apart
- Break out / delegate
  - New class Tab for the tab logic
  - openTab -> tab = new Tab
  - putOnTab -> tab.putOn
  - getTab -> tab.getSum
  - closeTab -> tab = null

# Design with Tab

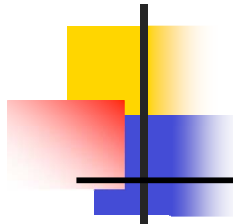




# Refactored Design

---

- Tab
  - Price calculation
  - Credit limit
  - (Cover drink-minimum)
- Still in DrinkSession EJB
  - Tab management
- Tab functionality easy to test
  - Money-grade PO JUnit Test + Mocked PriceList



# CODE!

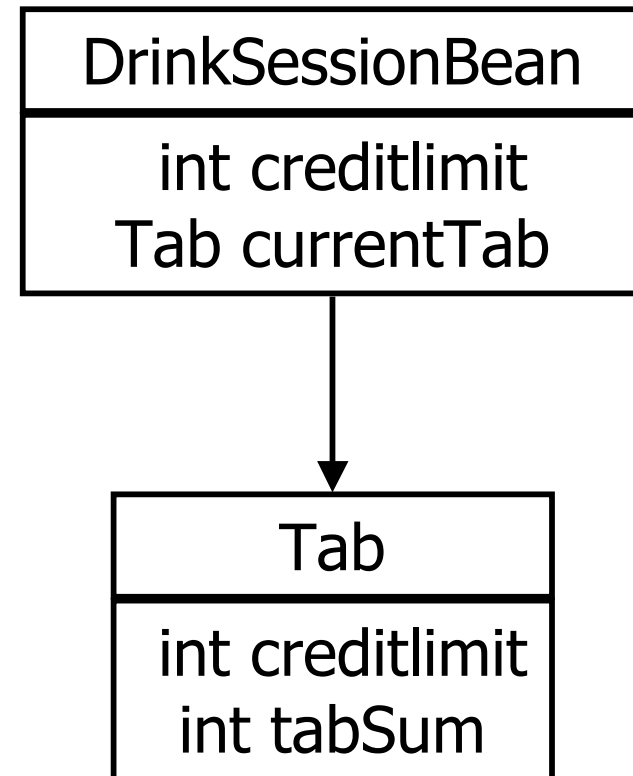
---

## DrinkSessionBean

- Remember credit limit
  - int creditlimit
- Keep track of tab or not
  - Tab currentTab

## Tab

- Keep track of sum
  - int tabSum
- Enforce credit limit
  - int creditlimit





# Evaluation

---

- More classes
  - Higher code complexity
  - Better coherence
  - Testable
    - Michael Feathers. "Before Clarity"
- Testable functions
  - Price calculation
  - Credit limit
  - But! Takes a mocked EJB-interface (PriceList)
- Still detestable
  - Tab management



# Principle Derived

---

- What we did:
  - Extracted business logic
  - Got rid of container complexity

Principle:

- ***Hard to get right – make easy to test***



# Scenario: Web Presentation

---

- Parts of page look different
  - Colour, Font, Read-only/Editable
- Logical condition based on
  - Data to show
  - State of presentation
  - Permissions, *etc.*
- Original requirements often simple
  - Conditions “creep in” as Change Requests



# Scenario: Web Presentation

---

## Result:

- Calculation of condition often end up in JSP
- JSP tend to become complex
  - Scriptlets
  - Logical tags
    - *e.g.* in Struts: `<logic:present>`, `<logic>equals>`
- Does the JSP really render correctly?
  - Need to test



# Example: Original Design

---

- Page showing tab info
  - Implementation: JSP tab.jsp
- Each order on its own line
- Data as list of OrderRowModel from Action
  
- Expensive orders should be in CAPITAL



# Problem and Solution

---

- JSP environment hard to test in
- JSP intended for View
- Our JSP contain Control
  - Condition calculation
  
- Make computation elsewhere
  - Struts Action
  - "set stage"
- JSP just rendering



# Refactored Design

---

- TabAction makes computation
- OrderRowModel contains formatted data
  - Or CSS class name (or another signal)
- JSP renders description “as is”



# Evaluation

---

- Same code elsewhere
- No Model bean?
  - Have to create it
  - More complexity
- No computations left in JSP
  - Easy to get right



# Principle Derived

---

- What we did:
  - Purified container component
  - Got rid of application logic

Principle:

- ***Hard to test – make easy to get right***



# Method

---

- Hard to ***get right*** – make easy to **test**
- Hard to **test** – make easy to ***get right***



# Scenario: Test of JSP

---

- Just rendering – unlikely to go wrong



# Example:

---

- JSP with rendering of order-lines
- One for loop
- Rendering of each line



# Evaluation

---

- No logic involved
  - Standard idioms
- “What could *possibly* go wrong?”
- Similar to get/set
  
- Too simple to fail
  - J. B. Rainsberger "JUnit Recipes"



# Principle Derived

---

- What we did
  - Somewhere we have to stop test our logic

Principle:

- ***Too Trivial to Test***



# Scenario: Correct Forward

---

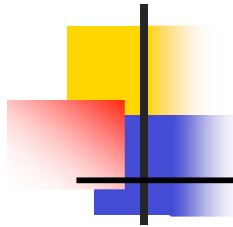
- Struts: input data in html form
- Action takes action
- Action decides forward
- Correct forward?
- Correct error message?



# Example:

---

- LoginForm with username and password
- Feed with valid pair
- Will we end up on “success”?
- Assume login validation well-tested



# CODE!

---

- LoginAction
- Test
  - Using StrutsTest



# Evaluation

---

- Everything is well tested on inside
- Do we need to test on outside?
- Just Struts inbetween
- Do we trust Struts?
  - Other frameworks?
  - Our application server?
  - Our database-driver?
- Should be well-tested
  - Or, add test to framework



# Principle Derived

---

- What we did:
  - We test our app
  - We do not test frameworks

Principle:

- ***Somebody Else's Problem***



# Method Endpoints

---

- ***Too Trivial to Test***
- ***Somebody Else's Problem***



# Method for Smarter Testing

---

## How to do it

- Hard to test – make easy to get right
- Hard to get right – make easy to test

## When to stop

- Too Trivial to Test
- Somebody Else's Problem



# Method Put to the Test

---

- Real world examples
- Take the method out for a test drive



# Challenge: Presentation State

---

- Tab shown on page
  - Expanded – all orders
  - Collapsed – just the sum
- Default expand
- Hidden field with info
- Buttons for expand/collapse



# Analysis

---

- State sent through cycle
  - HTML, ActionForm, JSP, HTML
  - Hard to test
- Advanced presentation state
  - Coupled to tab
  - Hard to get right



# Apply Method

---

- HTML, ActionForm, JSP
  - “Hard to test” environment
  - Make easy to get right
  - Push away state keeping
  - Put state in session
- Advanced presentation state
  - “Hard to get right” function
  - Make easy to test
  - Encapsulate in object



# Finished?

---

- Presentation state can be tested
- JSP just rendering
  - Too Trivial to Test
- Button will trigger Action
  - SEP
- Action update PresentationState
  - Too Trivial to Test



# Challenge: JNDI lookup

---

- JNDI lookup inside component



# Analysis

---

- JNDI linked to container
  - hard to test
- Looked up reference used in BL
  - hard to get right

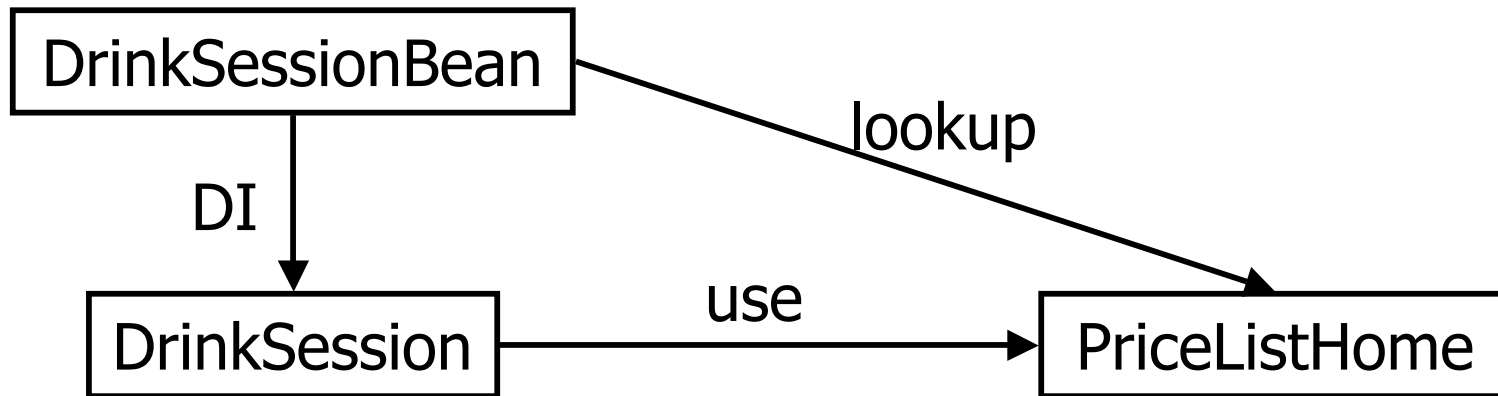
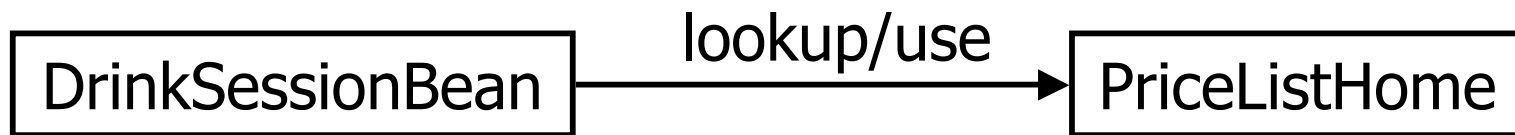


# Apply Method

---

- Logic separated to “logic object”
- JDNI lookup done in component
- Found reference passed in constructor / setter
- *aka* Dependency Injection

# Before and After





# Finished?

---

- Component perform lookup
  - SEP
- Component hands over reference
  - Too Trivial to Test
- Logic object uses reference
  - Testable and stubbable



# Challenge: JDBC

---

- Component use JDBC
- Fetch/save data from/to DB



# Analysis

---

- Database interaction
  - hard to test
- Data probably used in logic
  - Hard to get right



# Apply Method

---

- Separate JDBC to DAO
- Logic make call through interface



# Finished?

---

- DAO implementation can be tested
  - Well isolated
  - Perhaps Too Trivial to Test
- Use of data can be tested separately



# Challenge: EJB – Interdependencies

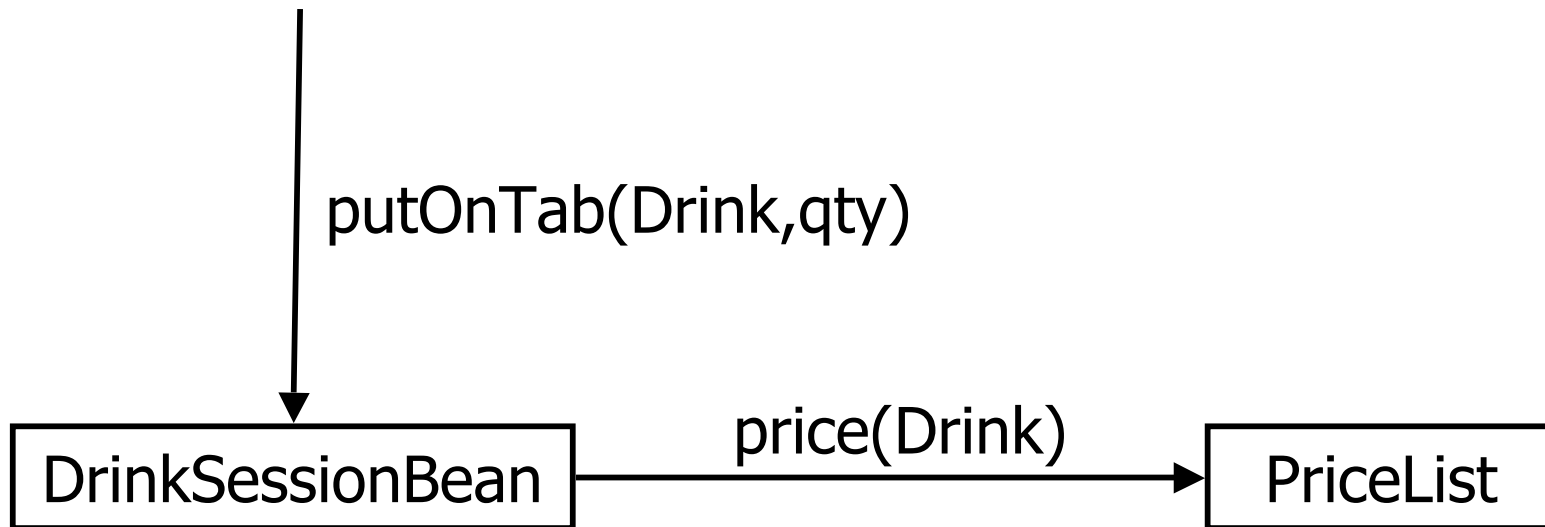
---

- EJB uses other EJB in BL

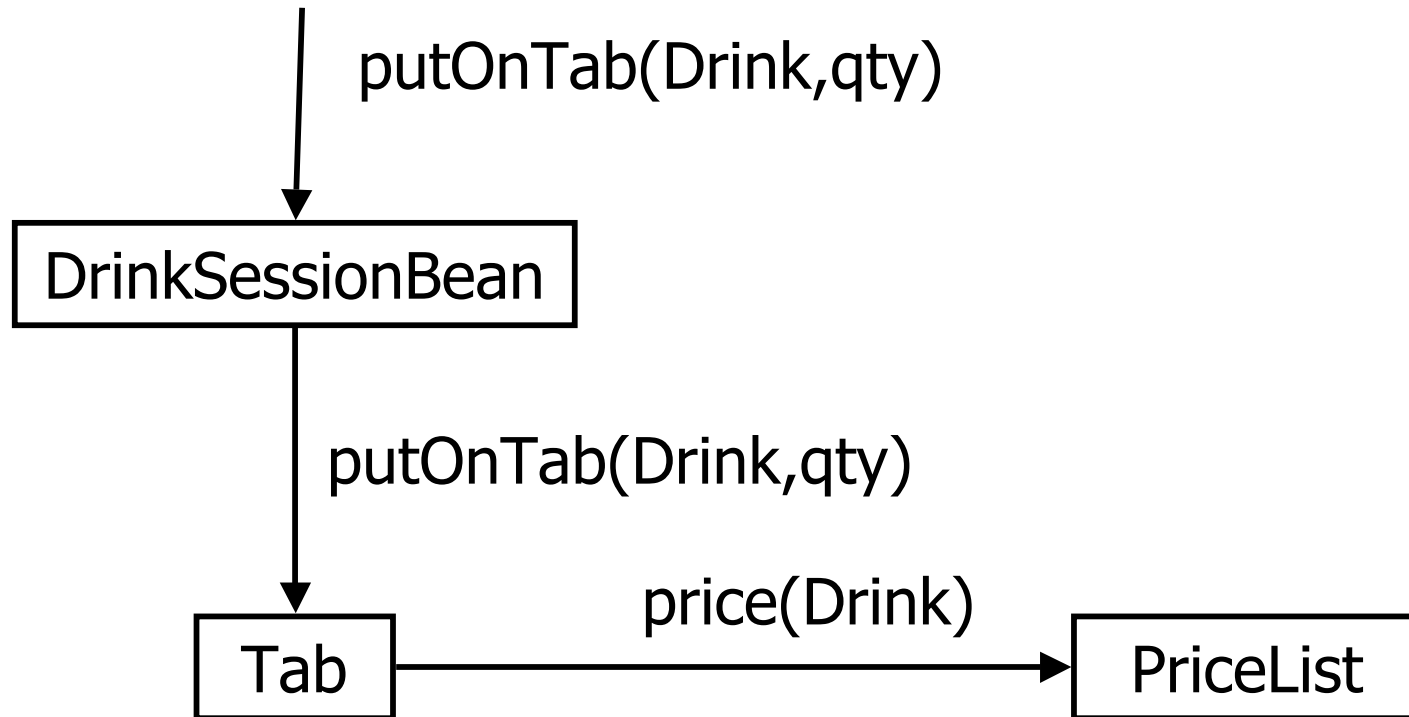


# Design

---



# Design with Tab





# Analysis

---

- Call to other EJB *via* container
  - Hard to test
- Use of call value in BL
  - Hard to get right

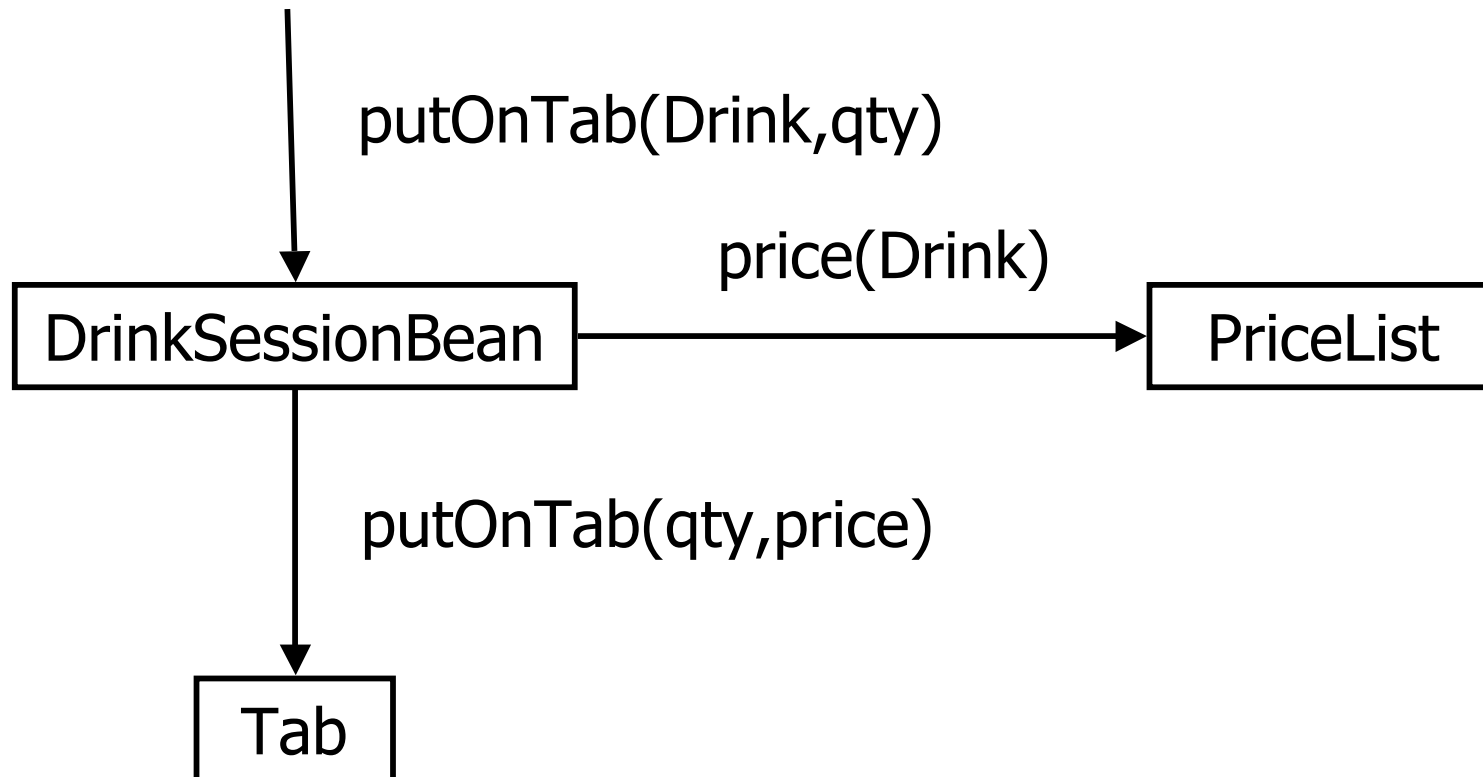


# Apply Method

---

- Separate fetch of value from use of value
- Use of value in BL object
- Fetch of value outside
- Value passed as parameter
- Tab is now a BL “dead end”

# Design with Tab





# Finished?

---

- Use of value can be tested
- Fetch of value *via* container
  - SEP
- Hand-over of value from call to BL object
  - Too Trivial to Test
  
- Move “out” from computation into “rigging”
- Put rigging outside of BL object



# Summary

---

- Focus on testing functionality/behaviour
- Do not always need complicated frameworks
- Simplify function in hard-to-test environment
- Move test-craving function to testable objects
- “Wiring” will be Too Trivial to Test
- Component behaviour often Somebody Else’s Problem



# Reminder

---

- Hard to get right – make easy to test
- Hard to test – make easy to get right



# References

---

- <http://strutstestcase.sourceforge.net/>
- Martin Fowler, *Refactoring*, ISBN: 0201485672  
[<http://www.martinfowler.com/bliki/DefinitionOfRefactoring.html>]
- Rod Johnson, *Expert One-on-One J2EE Development without EJB*, ISBN: 0764558315
- J. B. Rainsberger, *JUnit Recipes: Practical Methods for Programmer Testing*, ISBN: 1932394230
- Michael T. Nygard, "Test infect your Enterprise JavaBeans" *JavaWorld*, May 2000. [<http://www.javaworld.com/javaworld/jw-05-2000/jw-0526-testinfect.html>]
- Michael Feathers, "Before Clarity," *IEEE Software*, vol. 21, no. 6, pp. 86-88, November/December 2004.  
[<http://doi.ieeecomputersociety.org/10.1109/MS.2004.37>]
- Shakespeare, "Macbeth" 2.1  
[<http://www-tech.mit.edu/Shakespeare/macbeth/macbeth.2.1.html>]



# Effective Unit-Testing of In-Container Components

---

Dan Bergh Johnsson  
Omegapoint AB, Sweden

Hard to get right – make easy to test  
Hard to test – make easy to get right

