

Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ♦ twitter.com/pauloborba

To do before class

- Watch video
- Read chapters 8 and 11 in the textbook
- Send questions and opinions through slack

Design, implementation,
and maintenance

Model driven development (MDD)

Low level design and implementation

Should be guided by...

Values

Principles

Patterns

architectural, design, implementation

Coding values, Beck

- Communication: focus on the reader
- Simplicity: eliminate excess complexity
- Flexibility: easy to change, in expected ways
- Precision: names aligned to behavior

SOLID design principles

- A class should have a single responsibility (**single responsibility**)
- A module should be open for extension but closed for modification (**open closed**)
- Subclasses should be substitutable for their base classes (**Liskov Substitution**)
- Dependencies should be injected into a dependent object (**dependency injection**)
- A method of class C should only call methods of C or of the classes of C fields (**Demeter**)

More design principles

- Depend upon Abstractions, do not depend upon concretions (**dependency inversion**)
- Reusable code calls custom code, abstractions call concretions (**inversion of control**)
- Many client specific interfaces are better than one general purpose interface (**interface segregation**)

Package cohesion principles

- The granule of reuse is the granule of release (**release reuse equivalency**)
- Classes that change together, belong together (**common closure**)
- Classes that aren't reused together should not be grouped together (**common reuse principle**)

Package coupling principles

- The dependencies between packages must not form cycles (**acyclic dependencies**)
- Depend in the direction of stability (**stable dependencies**)
- Stable packages should be abstract packages (**stable abstractions**)

Other rules

Rule of Modularity: Write simple parts connected by clean interfaces.

Rule of Clarity: Clarity is better than cleverness.

Rule of Composition: Design programs to be connected to other programs.

Rule of Separation: Separate policy from mechanism; separate interfaces from engines.

Rule of Simplicity: Design for simplicity; add complexity only where you must.

Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.

Rule of Transparency: Design for visibility to make inspection and debugging easier.

Rule of Robustness: Robustness is the child of transparency and simplicity.

Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.

Other rules

Rule of Least Surprise: In interface design, always do the least surprising thing.

Rule of Silence: When a program has nothing surprising to say, it should say nothing.

Rule of Repair: When you must fail, fail noisily and as soon as possible.

Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.

Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.

Rule of Optimization: Prototype before polishing. Get it working before you optimize it.

Rule of Diversity: Distrust all claims for “one true way”.

Rule of Extensibility: Design for the future, because it will be here sooner than you think.

CAN YOU PASS
THE SALT?

I SAID—

I KNOW! I'M DEVELOPING
A SYSTEM TO PASS YOU
ARBITRARY CONDIMENTS.

IT'S BEEN 20
MINUTES!

IT'LL SAVE TIME
IN THE LONG RUN!

Patterns

- Context
- Problem
- Forces
- Solution
- Consequences
- Known uses

Professors and courses

| | | Disciplinas | | Professor | Professor |
|----------------|---|-------------|---|------------------------|-------------------------------|
| André Santos | Linguagem de programação 2 (com Hermo) | 38 | Física para computação | | |
| | Graduação em CC | 39 | Sistemas digitais | Marcel Lusob e de Lima | |
| | Compiladores | 40 | Lógica | Ruy Barreto de Queiroz | |
| | Graduação em CC | 41 | Metodologia e expressão escrita em TI | Paulina Tadeu | |
| | Paradigmas de linguagens computacionais | 42 | Informática e sociedade | Marcel Lusob e de Lima | |
| Edna Barros | Graduação em CC/IC | 43 | Estrutura de software | Carlos Fortes | Sérgio Cavalcante |
| | Teoria e implementação de linguagens de | 44 | Estrutura de hardware | Lidia Barros | |
| | Graduação em CC/IC | 45 | Estrutura de comunicação | Rajane Sadok | |
| | Carga: 1.5 | 46 | Processamento gráfico | Alexandro Orlandi de | Francisco Tenório de Carvalho |
| | | 47 | Interfaces usuário máquina | Alcides Gomes | |
| Edson Carvalho | Infra-estrutura de hardware | 48 | Gestão de dados e informações | Fernando Fonseca | |
| | Graduação em CC/IC | 49 | Sistemas inteligentes | Sérgio Hamill | Teresa Ludermir |
| | Arquitetura de computadores (com Sérgio) | 50 | Engenharia de software e sistemas | Alexandra Mota | |
| | Pós-graduação em CC | 51 | Processamento de desenvolvimento | Fernando Moura | Jacques Tassin |
| | Carga: 1.5 | 52 | Introdução à multimídia | Judith Kolner | |
| | Computador e sociedade | 53 | Paradigmas e linguagens computacionais | André Santos | Leandro Moura |
| | Graduação em CC | 54 | Teoria e implementação de linguagens computacionais | André Santos | |
| | Introdução à Informática | 55 | Informática teórica | Ruy Barreto de Queiroz | Alcides Cavalcante |
| | Turismo | 56 | História e futuro da computação | Sérgio Meira | |
| | Programação 1 (com Francisco Tenório de Carvalho) | 57 | Física para computação | | |



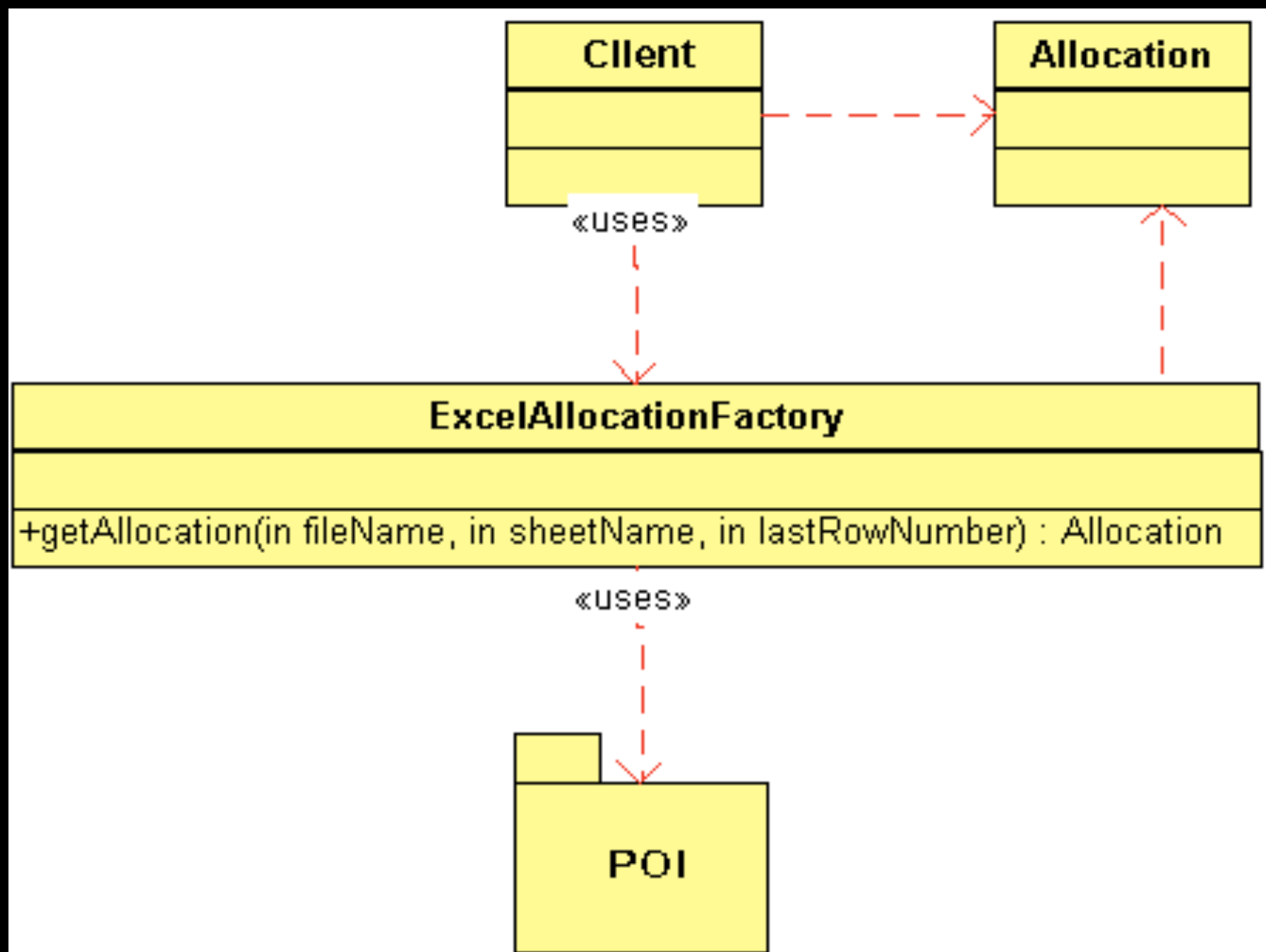
Java POI



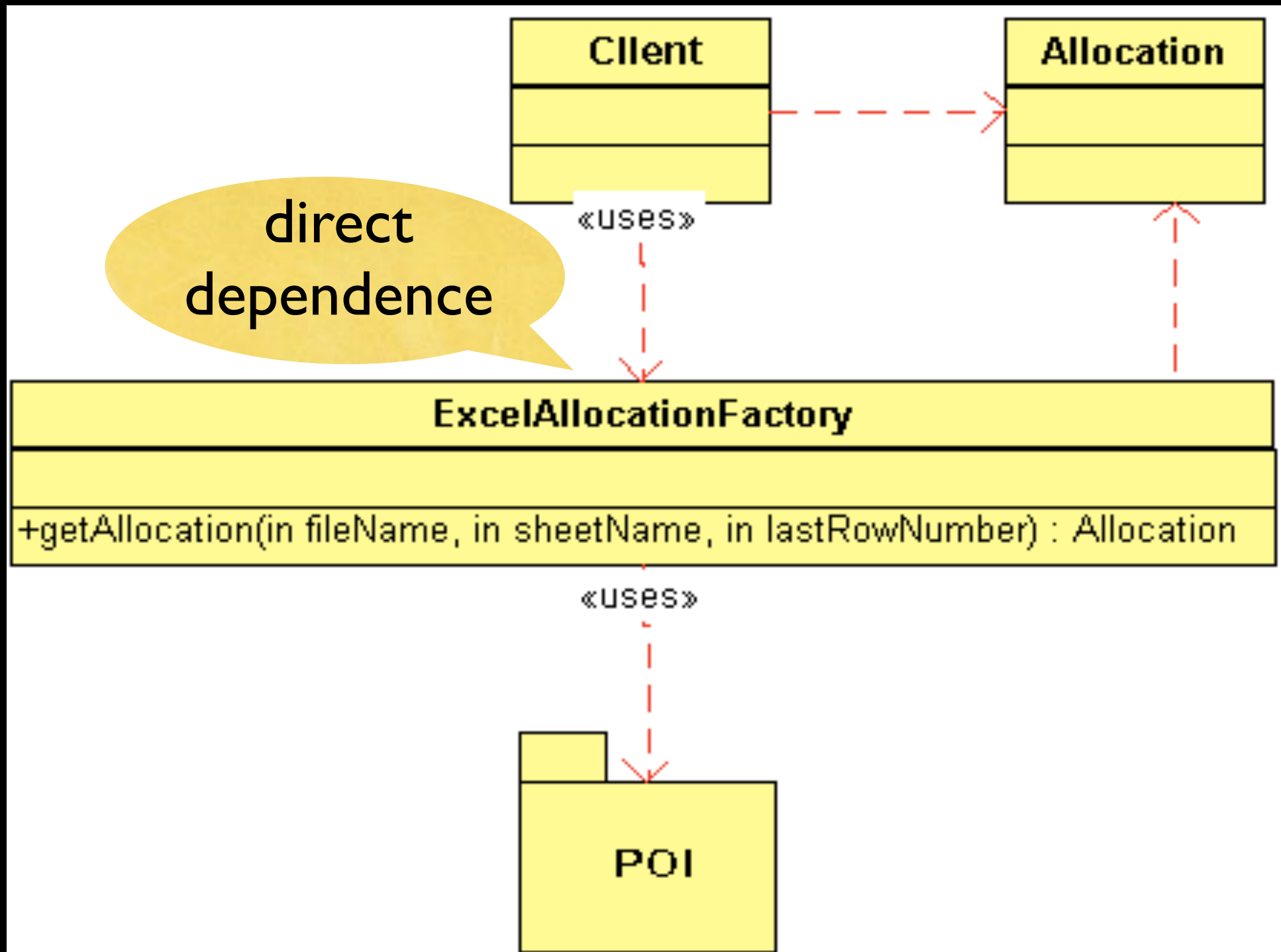
Java APIs Java for manipulating “OLE 2 Compound Document” files, such as **xls** e **doc**:

- HSSF (Horrible Spreadsheet Format)
- HDF (Horrible Document Format)
- HPSF (Horrible Property Set Format)

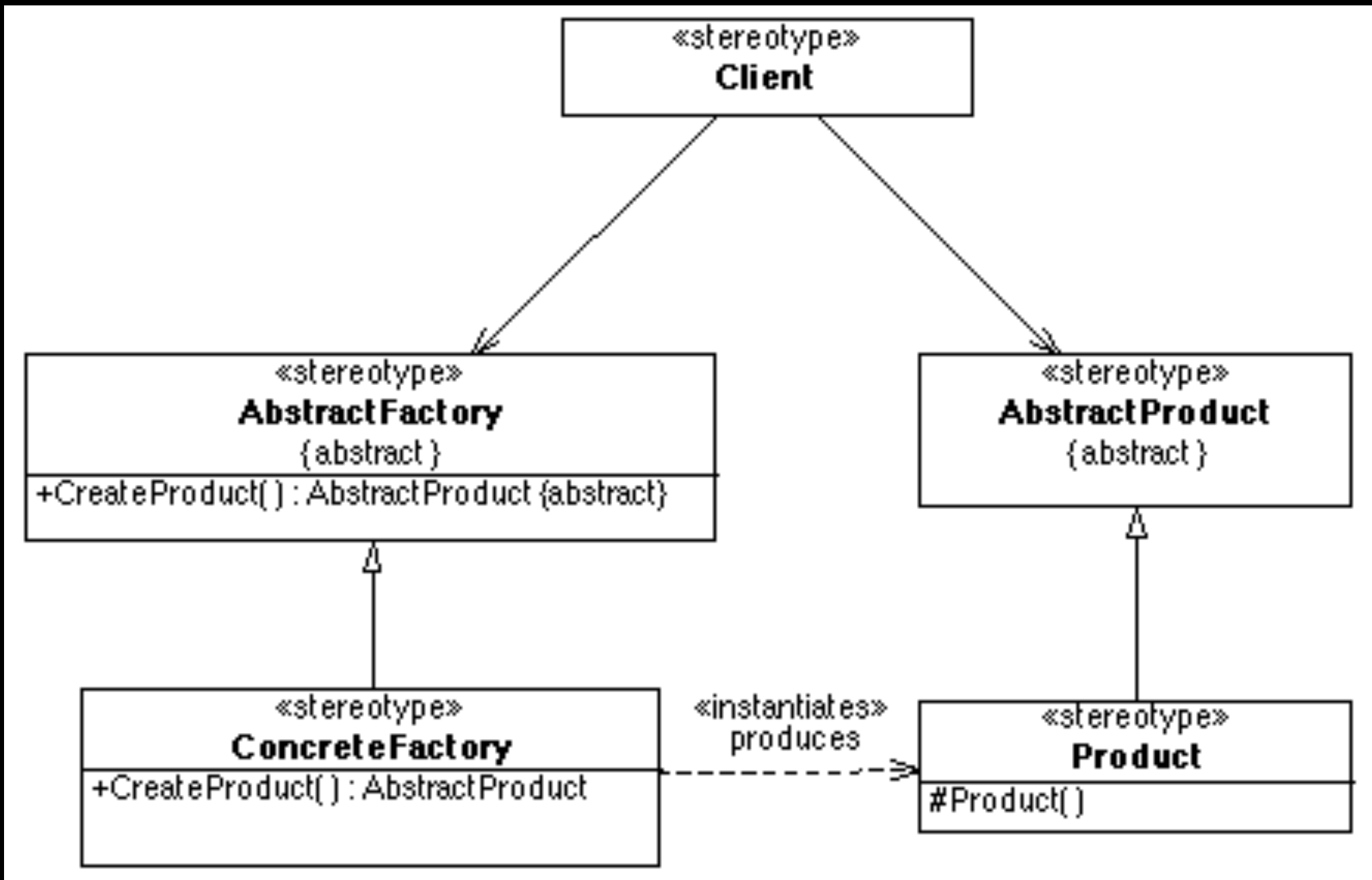
Factory pattern for using POI



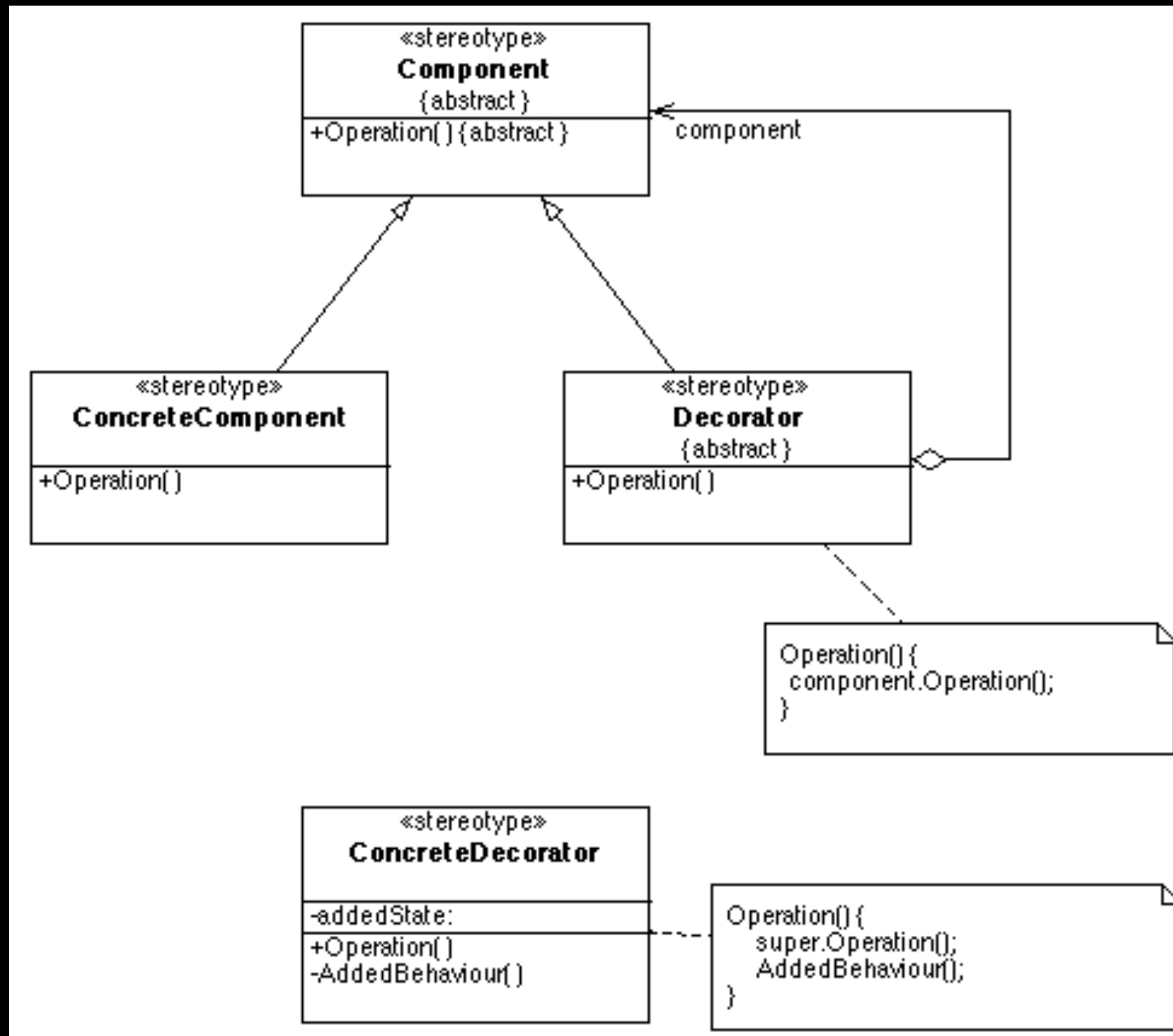
No interface?





Abstract factory




Decorator



Pattern catalogues


Wednesday, September 15, 2010 [Login](#) Text Size  

 THE HILLSIDE GROUP

Site Search

MAIN MENU








- ▶ Hilltop
- ▶ Conferences
- ▶ Patterns
 - Patterns Catalog
 - About Patterns
 - Books
 - TPLoP
 - Education
 - Mailing Lists
 - Writing
 - Tools
 - Links



Patterns > Patterns Catalog

DESIGN PATTERNS CATALOG

Display #

| # | Web Link | Hits |
|---|---|------|
| 1 |  Portland Pattern Repository maintained by Ward Cunningham. | 5703 |
| 2 |  Patterns of Interaction This page provides links to the patterns arising out of this project. Ultimately, we would hope for this to build into a resource for designers to draw upon when thinking about the requirements for cooperative systems. | 2371 |
| 3 |  EAM Pattern Catalog Wiki The objective of the EAM Pattern Catalog is to complement existing Enterprise Architecture (EA) management frameworks, which provide a holistic and generic view on the problem of EA management, and to provide additional detail and guidance needed to systematically establish EA management in a step-wise fashion within an enterprise. | 1270 |
| 4 |  Interaction Design Patterns This page contains information about resources related to pattern languages for interaction design (of which user interface design is a subset), and a few links to more general papers that may be of use to interaction designers. | 2362 |
| 5 |  System Reengineering Patterns Links to online patterns and papers. | 1021 |
| 6 |  GOF UML models | 3030 |
| 7 |  Workflow Patterns This site serves as a repository for workflow modelling patterns. | 1613 |

<http://hillside.net/patterns/patterns-catalog>

Strategy

- Analyze change request
- Any pattern for the given context?
- New solution based on values and principles?
- Apply appropriate pattern or principles
- Check whether new functionality breaks existing tests

For now, focus on bad **design** smells, related to classes and their relationships

Later we will focus on bad **code** smells, related to method implementations

Applies for
testing code
too!

Checklist

- Design and implementation conforms to discussed principles and patterns

Take notes,
now!

Design, implementation and evolution research at CIn

- Software product lines and evolution:
Leopoldo, Paulo
- Formal specification methods: Augusto,
Alexandre Mota, Juliano, Henrique, Márcio
- Concurrent systems and green design:
Fernando Castor
- Distributed systems architecture: Vinicius, Kiev

Hands-on!
Check assignment

To do after class

- Answer questionnaire (check classroom assignment), study correct answers
- Finish exercise (check classroom assignment), study correct answers
- Read, again, chapters 8 and 11 in the textbook
- Evaluate classes (check classroom assignment)
- Study questions from previous exams

To do after class, optional

- Estudar material sobre princípios de projeto
- Estudar princípios de codificação no Capítulo 3 do livro Implementation patterns, Kent Beck, 2008 (slides úteis para ter uma visão geral do livro)
- Navegar pelo catálogo de padrões

Questions from previous exams

- Descreva o princípio “Dependency Injection”. Explique as vantagens de seguir este princípio.
- Descreva o objetivo do padrão de projeto “Adapter”, e desenhe um diagrama de classes representando a sua estrutura.
- O princípio “Acyclic dependencies” estabelece que “The dependencies between packages must not form cycles”. Explique as vantagens de seguir esse princípio.
- Desenhe um diagrama de classes que ilustre o padrão de projeto “Decorator”, e explique as suas vantagens.

Questions from previous exams

- Descreva o princípio “Single Responsibility Principle”. Explique as vantagens de seguir este princípio.
- Descreva o princípio de projeto “Demeter”. Explique as vantagens de seguir este princípio.
- Desenhe um diagrama de classes que ilustre o padrão de projeto “Abstract Factory”.

Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ♦ twitter.com/pauloborba