

# Chapter 20 – MAPPING DESIGNS TO CODE

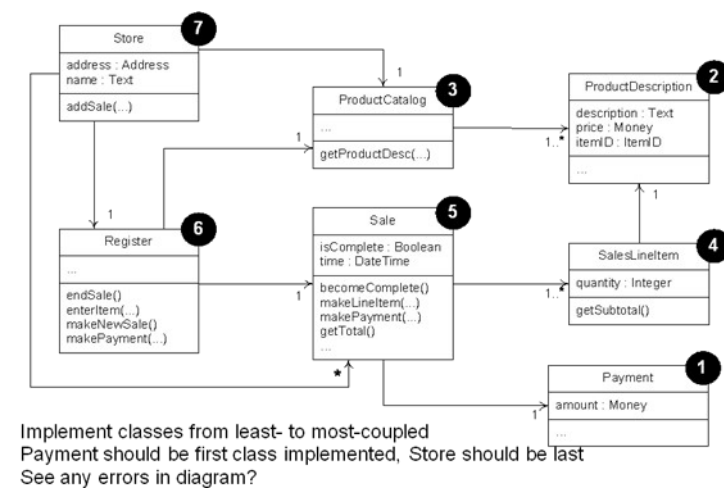
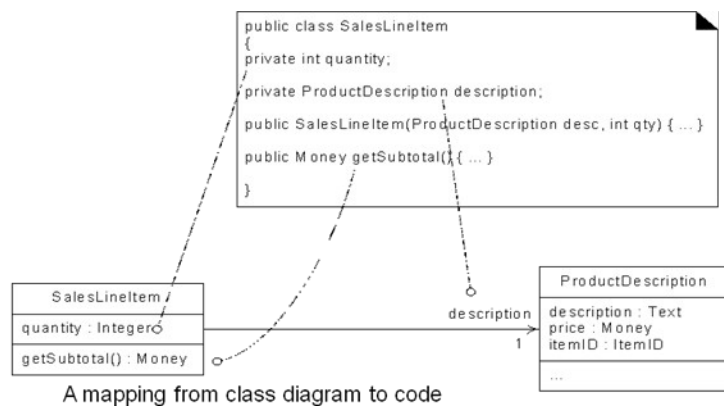
**Implementation Model** → The ultimate deliverables such as Source code, DB definitions, HTML pages etc.

In Design -> code is more than mechanical code generation

- Especially with agile modeling because many details skipped intentionally
- Developers may have good ideas that improve design

Implementation in OO language requires writing source code for:

- Definitions of classes & interfaces: get these from design class diagram. There are many tools can generate them.
- Definitions of methods



# Chapter 21 – TEST-DRIVEN DEVELOPMENT & REFACTORING

**Unit testing** → testing individual components.

**Test-Driven Development TDD or Test-First Development TFD** → test code is written before the class to be tested, and the developer writes unit-testing code for nearly all production code.

The basic rhythm is to write a little test code, then a little production code, make it pass the test, then write some more test code and so forth.

- Tests get written (not postponed)
  - Programmers get better at writing tests
  - Deeper understanding prior to coding
  - Reusable automated verification:
    - Rerun all unit tests after every code change
- \* Change code w/o excessive debugging

Note: JUnit is the testing framework for Java.

**Unit testing steps** →

- [1] Create object to be tested
- [2] Do something to it (invoke some method)
- [3] Check if results are correct

Write tests for 1 method at a time

**Refactoring** → is a structured, disciplined method of rewriting/restructuring existing code w/o changing its behavior, applying small transformation steps combined with re-executing tests each step.

- 'Refactoring: Improving the Design of Existing Code', by Fowler is must read
- Eclipse provides built-in support

**What are the activities and goals refactoring? Improve Design!**

- [1] Remove duplicate code
- [2] Improve clarity
- [3] Shorten methods
- [4] Remove hard-coded constants
- [5] Move methods to another class
- [6] ...and so on

**Code Smells** → is a metaphor in refactoring – they are hints that sth may be wrong in the code. “Look into the smelly code; it might turn out to be alright and not need improvement. In contrast,

**Code stench** → truly putrid code crying out for clean up.

**Some indicators:**

- [1] Duplicated code
- [2] Big methods
- [3] Class with many instance variables
- [4] Very similar subclasses
- [5] Little or no use of interfaces
- [6] High coupling

# Chapter 22 – UML Tools

Three ways to apply UML: as a sketch, as blueprint, and as a programming language.

In **CASE** (Computer Aided Sw Engineering):

Forward engineering → means the generation of code from diagrams.

**Reverse engineering** → means the generation of diagrams from code.

**Round-trip Engineering** → ‘closes the loop’: the tools support generation in either direction and can synchronous between UML diagrams and code.

## Chapter 23/24 – ITERATION 2

*When Iteration-1 ends, discuss what normally should be accomplished?*

**By end of iteration 1** →

- [1] All the SW has been thoroughly tested: unit, acceptance, load, usability and so on.
- [2] The idea of UP is to do early, realistic, and continuous verification of the quality and correctness of the system. Thus, customers(users) have been regularly engaged in evaluating the partial system. Feedback from users guide to developers for adaptation and clarification of requirements.
- [3] The System, across all the subsystems, has been completely integrated and stabilized.

**Activities concluding Iteration-1** →

- [1] An Iteration planning meeting to decide what to work on the next iteration.
- [2] At the start of the new iteration, use UML tool to reverse engineer class diagram
- [3] Usability analysis for the UI is underway.
- [4] DB modeling & implementation is underway.
- [5] Do a requirements workshop: Pick UC scenarios for next iteration

*Discuss briefly what are the focuses of Iteration-2?*

**Elaboration: Iteration-2**

**Iteration-2 Requirements** →

**Iteration-2 Analysis** →

**Iteration-2 GRASP: more obj. with responsibilities** →

**Iteration-2 Applying GOF design patterns**

- [1] Iteration-2 largely ignores requirements analysis.. but it includes **Incremental development for UCs across iterations.**
- [2] Iteration-2 includes quick **analysis update**: domain analysis, SSDs and DM refinement. DM now may have **Generalization-Specialization (or class hierarchy)** → i.e., Inheritance!

Gen. is the activity of identifying commonality among concepts and defining superclass (general concept) and subclass (specialized concept) relationship.

**Model subclasses when they**

1. Have additional attributes,
2. Have additional associations
3. Are manipulated or behave in a different manner a.i.e., methods have different implementations

[3] Iteration-2 focuses on **obj. design with responsibilities and GRASP,**

[4] Iteration-2 applies some **GOF design patterns.**