Chapter 16: Design Class Diagrams (DCD)

GRASP: General Responsibility Assignment Software Patterns (Principles)

- Creator
- Information Expert
- Low Coupling
- Controller
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

Chapter 19: Designing for Visibility

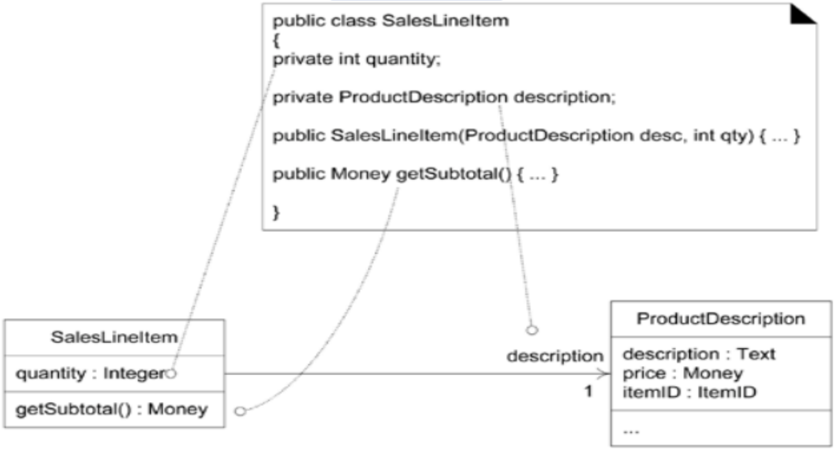
- What Is Visibility: The ability for an object to “see” or reference another object, related to the issue of scope. Is one resource within the scope of another?
  - Attribute Visibility:
    - B is an attribute of A
    - Permanent visibility
  - Parameter Visibility:
    - B is a parameter of Method A
    - Temporary visibility, only within scope of the method
  - Local Visibility:
    - B is a (non-parameter) local object in a method of A
    - Temporary visibility, only within scope of method
  - Global Visibility:
    - B is in some way globally visible
    - Relatively permanent visibility

Chapter 20: Mapping Designs to Code

- Creating Class Definitions from a DCD

Figure 20.1. SalesLineItem in Java.

[View full size image]

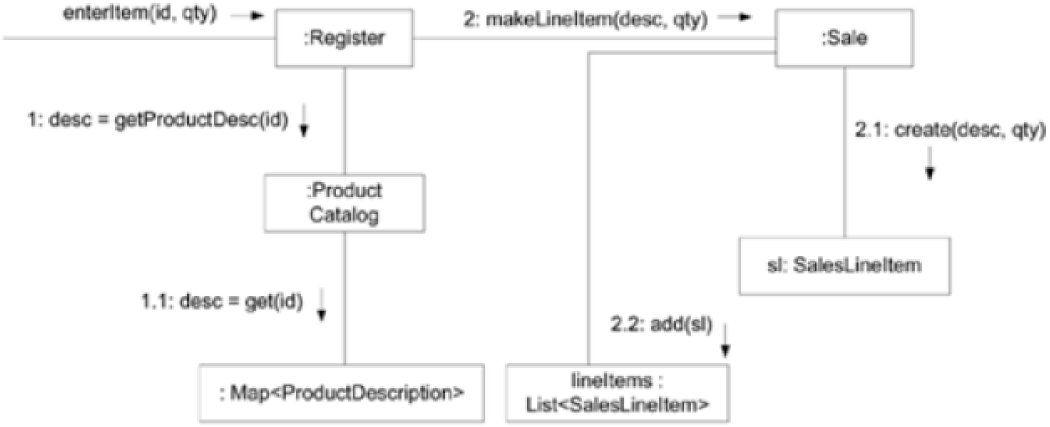


```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

- Creating Methods from Interaction Diagrams

Figure 20.2. The enterItem interaction diagram.

[View full size image]



```
public void enterItem(ItemID itemID, int qty)
{
        //Message 1
        ProductDescription desc = catalog.getProductDescription(itemID);
        //Message 2
        currentSale.makeLineItem(desc, qty);
}
```

Chapter 21: Test Driven Development and Refactoring

- test-driven development (TDD) or test-first development. In this practice, unit testing code is written before the code to be tested, and the developer writes unit testing code for all production code. The basic rhythm is to write a little test code, then write a little production code, make it pass the test, then write some more test code, and so forth.

- TDD
  - Unit tests get written
  - Consistent test writing
  - Clarification about detailed behavior or interfaces
  - Provable, repeated, automated verification
  - Confidence to change errors

- Refactoring: Changing the code without changing external behavior
  - Remove duplicate code
  - Reduce size of methods
  - Improve clarity

Iterator Pattern