Chapter 12. Addition, Finally

Testing and Development Approach

Test-Driven Development (TDD)
- Start with simple tests and fake implementations
- Gradually refactor towards real implementation
- Use tests to guide design and gain confidence

Progress Summary
- Reduced complex test to simpler one ($5 + $5)
- Chose metaphors for computation
- Rewrote tests based on metaphors
- Achieved compilation and passing tests with stubbed methods
- Anticipate refactoring for real implementation

Bank Responsibilities

Why Bank Handles Reduction
- Keeps Expression objects simple and flexible
- Avoids Expression class growing too complex
- Willing to shift responsibility if Bank not needed

Bank API
- Method: Money reduce(Expression source, String to)
- Initially stubbed to return Money.dollar(10)

Simple Addition Example

Basic Arithmetic
- $5 + $5 = $10
- Test example: testSimpleAddition()

Implementation
- Money.plus(Money addend) returns new Money(amount + addend.amount, currency)
- Money class implements Expression interface

Multi-Currency Arithmetic

Design Constraints
- Most code should be unaware of multiple currencies
- Instant conversion to reference currency (e.g., USD) is inflexible for variable exchange rates

Desired Solution
- Represent multiple exchange rates conveniently
- Arithmetic expressions should resemble natural arithmetic

Use of Objects and Expressions

Concept of Imposter Object
- Create object acting like Money but representing sum of two Moneys

Metaphors
- Wallet: holds multiple notes of different denominations and currencies
- Expression: operations like "(2 + 3) * 5" or "($2 + 3 CHF) * 5"
- Money is atomic expression
- Operations produce Expressions including Sum

Expression Reduction
- Expressions reduced to single currency given exchange rates
- Reduction performed by Bank