Developing a Programming Aesthetic

- Summary of Precepts for Programming Aesthetic
  - Put domain behavior on instances
  - Avoid instance methods knowing concrete class names
  - Depend on injected abstractions, not hard-coded concretions
  - Push object creation to application edges, separate creation from use
  - Avoid Law of Demeter violations; use violations as clues for deeper abstractions
  - These principles loosen coupling, increasing adaptability and maintainability
  - Accept short-term complexity for long-term flexibility and easier change
  - Develop internal heuristics (aesthetic) to guide decisions under uncertainty
- Pushing Object Creation to the Edge
  - BottleVerse#lyrics instance method still depends on concrete BottleNumber class
  - Object creation (BottleNumber.for(number)) should be moved to outer layers (edges)
  - Modify BottleVerse.lyrics class method to convert number to BottleNumber before creating instance
  - Refactor BottleVerse to accept BottleNumber instance instead of number
  - Gradually remove redundant number references with one-undo refactoring steps
  - Final BottleVerse class:
    - class method .lyrics(number) creates instance with BottleNumber.for(number)
    - instance method #lyrics uses injected bottle_number without knowing creation details
- Applying LoD to Bottles#verse
  - Current code verse_template.new(number).lyrics violates LoD (message chain on injected class)
  - Rule: inject instances, not classes, to avoid chaining new message
  - Can't inject instance because number varies per verse
  - Solution: add class method BottleVerse.lyrics(number) that creates instance internally and forwards lyrics call
  - Bottles#verse simplified to verse_template.lyrics(number) obeying LoD
- Law of Demeter (LoD)
  - Understanding the Law
    - LoD restricts message sending to:
      - objects passed as method arguments
      - objects directly held by self
    - Avoid chaining calls to collaborators' collaborators (e.g., best_friend.pet.preferred_toy.durability)
    - Violations cause tight coupling, hard to reuse and test, and obscure intention
  - Curing Demeter Violations
    - Use message forwarding (delegation) to replace chains with single messages
    - Rename messages to reflect caller's intent rather than concatenation of object roles
    - Example: Instead of best_friend.pet.preferred_toy.durability, Foo asks best_friend.playdate_time_limit
    - Enables easier reuse and adaptability
- Grappling with Inversion
  - DIP means high-level modules (Bottles) depend on abstractions (verse template role)
  - Removes dependency on concrete classes (BottleVerse)
  - Injection loosens coupling and supports polymorphism and extensibility
- Mechanical Process Appreciation
  - "99 Bottles" code evolved by formal refactorings and recipes
  - Refactorings guided by code smells and recipes limit unnecessary tinkering
  - Working code may feel unsatisfying due to duplication or maintainability fears
  - Voluntary changes cost time and opportunity cost must be considered
  - Judgement and intuition guide when to improve code beyond recipes
- Programming Aesthetic and Intuition
  - Experienced programmers develop an aesthetic sense for code quality
  - Intuition highlights potential improvements but must be verbally justified
  - Programming aesthetic = principles guiding decisions under uncertainty
- Clarifying Responsibilities with Pseudocode
  - New requirement: support songs similar to "99 Bottles" with different lyrics
  - Current code counts down verses; not open to varying verse generation
  - Adding conditionals to verse method introduces code smells (Switch Statement, Long Function)
  - Better to extract each variant into its own class (Extract Class)
- Extracting the Verse
  - Extract BottleVerse class from Bottles to isolate verse responsibility
  - Steps:
    - Create BottleVerse class with initialize and attr_reader for number
    - Copy verse method to BottleVerse
    - Modify Bottles#verse to delegate to BottleVerse
    - Run tests to verify correctness
  - Initial extraction leads to redundant parameters which can be cleaned up later
- Coding by Wishful Thinking
  - Alternative to incremental refactoring: code by speculation
  - Replace Bottles#verse with BottleVerse.new(number).lyrics to express intent
  - Rename BottleVerse#verse to BottleVerse#lyrics
  - This approach reduces refactoring steps but adds risk; tests must confirm correctness
  - Encourages developing programming aesthetic by trying different techniques
- Inverting Dependencies (Dependency Inversion Principle - DIP)
  - Bottles depends directly on concrete BottleVerse class, causing tight coupling
  - DIP: depend on abstractions, not concretions
  - Inject verse_template (default to BottleVerse) into Bottles to decouple
  - Bottles then delegates verse generation to any injected verse_template
  - Enables easily swapping different verse templates without changing Bottles
- Isolating Variants
  - Identify code to vary, name concept, extract code to class, inject back, forward messages
  - Bottles composed of verse_template role players
  - Allows creating and injecting new verse templates for new lyrics variants