Manufacturing Intelligence

Summary

- Maintainable OO code depends on polymorphism and small interchangeable role players
- Variants encapsulate behavior, removing conditionals from within classes
- Factories isolate conditionals selecting correct variants in one place
- Various factory designs trade off openness, complexity, and knowledge ownership
- Effective factories enable polymorphism and improve code flexibility and maintainability

Polymorphism and BottleNumber Classes

- Instances polymorphically play the role of bottle number
- Each class implements a variant without conditionals
- Conditional logic still needed to select correct class
- Selection happens in BottleNumber.for factory

Auto-registering Candidates

- Using inherited Hook
  - Ruby sends inherited(candidate) message to superclass when subclassed
  - Factory overrides inherited to register subclass automatically
  - Removes need for explicit registration in subclasses
- Registry Initialization
  - Initialize registry to include BottleNumber class as default
  - Ensures default role player is present in registry
- Benefits and Trade-offs
  - Candidates registered automatically on definition
  - Requires all candidates inherit from factory class
  - Efficient and allows any class naming

Contrasting Concrete Factory with Shameless Green

- Replace Conditional with Polymorphism
  - Small bottle number hierarchy open for extension
  - Adding new behavior by adding new BottleNumber subclass
  - Factory still contains hard-coded conditional selecting class
- Factory Example (Case Statement)
  - Case statement chooses class based on number
  - Conditional logic remains centralized in factory
- Shameless Green Verse Method
  - Uses conditional to produce verses including behavior
  - Contains more branches due to verse-specific logic
- Key Differences
  - Shameless Green conditional contains behavior logic
  - Factory conditional selects an object/class
  - Factories separate choosing from behavior

Self-registering Candidates

- Registry Pattern
  - Factory holds a registry array of candidate classes
  - Provide register(candidate) method to prepend candidates
  - Candidates register themselves by calling BottleNumber.register(self)
- Benefits
  - Removes hard-coded candidate list from factory
  - Factory uses registry to find suitable candidate
- Dependency Considerations
  - Explicit registration via BottleNumber.register(self) depends on factory name stability
  - Implicit registration via inheritance depends on class hierarchy stability
  - Choice depends on which is more stable in project context

Understanding Factories

- Role of Polymorphism
  - Objects play interchangeable roles with common API
  - Message senders depend on interface, not implementation details
  - Allows system to be tolerant of unexpected change
- Role of Factory
  - Knows how to select correct variant for a situation
  - Contains conditionals isolated in one place
  - Factories "where conditionals go to die"
- Dimensions of Factories
  - Open or closed to new variants
  - Factory or variant owns choosing logic
  - Factory knows eligible classes or variants volunteer themselves

Dispersing the Choosing Logic

- Logic Owned by Variants
  - Each candidate class implements handles?(number) to decide if it handles the number
  - Factory iterates over candidates and selects first that responds true
- Benefits
  - Choosing logic co-located with relevant class
  - Useful if choosing logic is complex or changes with class
- Issues
  - Factory still has hard-coded list of candidates (closed)
  - Order matters as default handles? returns true always, must be last
  - Potential conflicts if multiple candidates respond true

Opening the Factory (Open for Extension)

- Naming Convention and Meta-Programming
  - BottleNumber classes follow naming convention: BottleNumber0, BottleNumber1, etc.
  - Factory dynamically derives class name using const_get and rescues NameError
- Benefits and Objections
  - Open to extension without changing factory code
  - Harder to understand, uses exceptions for flow control
  - Classes not explicitly referenced, risking undetected deletion
  - Silent failures if naming convention broken
- When to Use
  - Worthwhile if frequently adding new bottle number classes
  - Trade-off between complexity and cost of change

Supporting Arbitrary Class Names

- Key/Value Lookup Factory
  - Replace case statement with hash mapping numbers to classes
  - Hash defaults to BottleNumber if key missing
  - Allows arbitrary class names and centralized knowledge
- Comparison with Other Factories
  - Case statement is simple and readable but closed
  - Meta-programmed factory is open but requires naming convention
  - Key/value factory is open and allows arbitrary names but slightly harder to read
- Separation of Data and Algorithm
  - Hash stores "this->that" mapping; lookup logic separate
  - Allows externalizing data for configuration without code changes