

Achieving Openness in Code Refactoring

Overview

- Refactor code to meet six-pack requirement.
- BottleNumber class isolates changeable parts.
- Removing Data Clumps and conditionals improves maintainability.
- Uses Factory and polymorphism to handle variations.
- Fixes Liskov Substitution Principle violation.

6.1. Consolidating Data Clumps

- Problem Identification**
 - quantity and container methods often used together.
 - This pairing is a Data Clump code smell.
 - Data Clump: multiple data fields always appear together.
- Solution**
 - Extract concept by creating a method or class representing the clump.
 - Implement to_s method in BottleNumber to return "#{quantity} #{container}".
- Benefits**
 - Simplifies verse method by replacing quantity/container pairs with to_s.
 - Improves readability and reduces duplication.
- Example**
 - Verse method uses "#{bottle_number}" which calls to_s implicitly.
 - Capitalization moved to the sentence level, not on bottle_number.

6.2. Making Sense of Conditionals

- Problem**
 - BottleNumber class has many conditionals checking the number.
 - Conditionals represent Switch Statement smell.
- Background**
 - Previously, conditionals were in Bottles class; moved to BottleNumber class.
 - Conditionals depend on BottleNumber's number attribute.
- Refactoring Options**
 - Replace Conditional with State/Strategy (composition).
 - Replace Conditional with Polymorphism (inheritance).
- Choice**
 - Polymorphism chosen for this problem as it fits six-pack requirement.
- Skilled Programmer Insight**
 - They identify code smells and apply appropriate refactorings intuitively.
 - Experimentation and evaluation guide decisions.

6.3. Replacing Conditionals with Polymorphism

- Polymorphism Concept**
 - Different objects respond to same message.
 - Senders care about behavior, not object type.
- Dismembering Conditionals**
 - Identify special cases: numbers 0 and 1.
 - Extract subclasses BottleNumber0 and BottleNumber1.
 - Move conditional logic into subclasses.
- Manufacturing Objects with Factory**
 - Create factory method bottle_number_for in Bottles class to instantiate correct subclass.
 - Factory isolates class selection logic.
 - Refactor factory to minimize duplication.
- Implementation Details**
 - Factory returns class, then calls new(number).
 - Polymorphic methods like quantity, container, action, pronoun, successor implemented differently in subclasses.
- Prevailing with Polymorphism**
 - Repeat extraction for all methods with conditionals.
 - Add subclasses and update factory to handle new subclasses.
 - Example: BottleNumber0 overrides quantity, action, successor.
 - BottleNumber1 overrides container, pronoun.
- Code Structure**
 - Bottles class uses factory to get bottle numbers.
 - BottleNumber base class handles default behavior.
 - Subclasses handle special cases.
- Benefits**
 - Code clearly represents domain concepts.
 - Easier to maintain and extend.
 - Liskov violation discovered in successor method.

6.7. Summary

Achievements

Removed Data Clumps

6.6. Defending the Domain

- Importance of Domain Concepts**
 - quantity and container reflect fundamental domain ideas.
 - to_s should be generic and not encode domain-specific combinations.
- Risks of Shortcut**
 - Overriding to_s with domain-specific string breaks abstraction.
 - Leads to tight coupling and less reusable code.
 - Future programmers may misunderstand class responsibilities.
- Best Practice**
 - Maintain clear domain methods.
 - Let to_s use those methods to produce string representation.

6.5. Making the Easy Change (Adding Six-Pack)

- Opportunity**
 - Code now open to extension for six-pack requirement.
- Implementation**
 - Create BottleNumber6 subclass of BottleNumber.
 - Override container to return "six-pack".
 - Override quantity to return "1".
 - Update factory to return BottleNumber6 for number 6.
- Testing**
 - Update tests to expect "1 six-pack" for verses 6 and 7.
 - Tests fail until factory updated.
 - After update, tests pass, fulfilling six-pack requirement.
- Key Insight**
 - Adding a new subclass is easy after refactoring.
 - Highlights benefit of making change easy before making the change.
- Alternative (Not Recommended)**
 - Override to_s in BottleNumber6 to return "1 six-pack".
 - This couples BottleNumber6 to current usage context.
 - Hides domain concepts quantity and container.
 - Less reusable and misleading for future use.
- Recommendation**
 - Implement quantity and container explicitly to preserve domain clarity.

6.4. Transitioning Between Types (Fixing Liskov Violation)

- Problem**
 - successor method returns Integer instead of BottleNumber.
 - Violates Liskov Substitution Principle.
 - Causes increased dependencies and fragile code.
- Goal**
 - successor methods should return BottleNumber instances.
 - verse method expects successor to return BottleNumber.
- Strategy**
 - Move factory method to BottleNumber class as class method for.
 - Modify successor methods to use factory to return BottleNumber.
- Transitional Steps**
 - Add guard clause in factory to return object if already BottleNumber.
 - Modify BottleNumber0 and BottleNumber successor methods to call BottleNumber.for.
 - Update verse method to use successor directly.
 - Inline temporary variables in verse method.
 - Remove guard clause after transition complete.
- Result**
 - All successor methods return BottleNumber.
 - verse method simplified and trustful of successor.
 - Code adheres to Liskov Substitution Principle.
 - Trustworthy objects simplify interactions.
- Importance**
 - Avoid type checks and conditionals scattered in code.
 - Polymorphism enables flexible, maintainable design.