Java by Comparison Kata

github.com/javabycomparison/kata

Team:

1. Start Cleaning Up

- [ ] Avoid Unnecessary Comparisons
- [ ] Avoid Negations
- [ ] Return Boolean Expressions Directly
- [ ] Simplify Boolean Expressions
- [ ] Avoid NullPointerException in Conditionals
- [ ] Avoid Switch Fallthrough
- [ ] Always Use Braces
- [ ] Ensure Code Symmetry

2. Level Up Your Code Style

- [ ] Replace Magic Numbers with Constants
- [ ] Favor Enums Over Integer Constants
- [ ] Favor For-Each Over For Loops
- [ ] Avoid Collection Modification During Iteration
- [ ] Avoid Compute-Intense Operations During Iteration
- [ ] Group with New Lines
- [ ] Favor Format Over Concatenation
- [ ] Favor Java API Over DIY

3. Use Comments Wisely

- [ ] Remove Superfluous Comments
- [ ] Remove Commented-Out Code
- [ ] Replace Comments with Constants
- [ ] Replace Comments with Utility Methods
- [ ] Document Implementation Decisions
- [ ] Document Using Examples
- [ ] Structure JavaDoc of Packages
- [ ] Structure JavaDoc of Classes and Interfaces
- [ ] Structure JavaDoc of Methods
- [ ] Structure JavaDoc of Constructors

4. Name Things Right

- [ ] Use Java Naming Conventions
- [ ] Follow Getter/Setter Conventions for Frameworks
- [ ] Avoid Single-Letter Names
- [ ] Avoid Abbreviations
- [ ] Avoid Meaningless Terms
- [ ] Use Domain Terminology

5. Prepare for Things Going Wrong

- [ ] Fail Fast
- [ ] Always Catch Most Specific Exception
- [ ] Explain Cause in Message
- [ ] Avoid Breaking the Cause Chain
- [ ] Expose Cause in Variable
- [ ] Always Check Type Before Cast
- [ ] Always Close Resources
- [ ] Always Close Multiple Resources
- [ ] Explain Empty Catch

6. Assert Things Going Right

- [ ] Structure Tests Into Given-When-Then
- [ ] Use Meaningful Assertions
- [ ] Expected Before Actual Value
- [ ] Use Reasonable Tolerance Values
- [ ] Let JUnit Handle Exceptions
- [ ] Describe Your Tests
- [ ] Favor Standalone Tests
- [ ] Parametrize Your Tests
- [ ] Cover the Edge Cases

7. Design Your Objects

- [ ] Split Method with Boolean Parameters
- [ ] Split Method with Optional Parameters
- [ ] Favor Abstract Over Concrete Types
- [ ] Favor Immutable Over Mutable State
- [ ] Combine State and Behavior
- [ ] Avoid Leaking References
- [ ] Avoid Returning Null

8. Let Your Data Flow

- [ ] Favor Lambdas Over Anonymous Classes
- [ ] Favor Functional Over Imperative Style
- [ ] Favor Method References Over Lambdas
- [ ] Avoid Side Effects
- [ ] Use Collect for Terminating Complex Streams
- [ ] Avoid Exceptions in Streams
- [ ] Favor Optional Over Null
- [ ] Avoid Optional Fields or Parameters
- [ ] Use Optionals as Streams