

Java 言語で学ぶリファクタリング入門

0 章 リファクタリングとは

著者: 結城浩

発表者: 野口耕二郎

目次

- リファクタリングとは
- リファクタリングと「不吉な匂い」
- リファクタリング・カタログ
- リファクタリングのエッセンス
- リファクタリング Q&A
- まとめ・所感

リファクタリングとは

リファクタリングの定義

- 外部から見た振る舞いを変えずに, プログラムの内部構造を改善すること
- <クラスの抽出>というリファクタリングでは
 - 内部で巨大なクラスが分割されて構造が改善した
 - 外部から見るとプログラムの振る舞いは変わっていない

リファクタリングとユニットテスト

- 振る舞いが本当に変化していないことを確かめるために, 少なくともユニットテストを行う必要がある
- Java の場合は JUnit を用いる
- 実際の手順は
 - リファクタリングを行う前にテストを行う
 - リファクタリングを行う
 - リファクタリングをしたあとに再度テストを行う
 - 前後のテストの結果が一致していることを確かめる

リファクタリングの目的

- バグを見つけやすくする
 - プログラムが整理されて潜んでいるバグを見つけやすくなる
- 機能追加しやすくする
 - 機能追加するとソースコードは複雑になる
 - 汚いコードを整えることで機能追加が楽になる
- レビューしやすくする
 - コードが理解しやすくなりレビューの速度や精度が上がる

リファクタリングの限界

- プログラムがまだ動いていない場合
 - コンパイルエラーなどが発生している場合はテストできない
- あまりにも時間がない場合
 - リファクタリングを失敗すると新たなバグが混入する
 - そもそもリファクタリングは時間的に少し後からじわじわと効いてくる

リファクタリングと「不吉な匂い」

不吉な匂いとは

- プログラムの「理解しにくい」、「修正しにくい」、「拡張しにくい」と感じる部分のこと
- 書籍『リファクタリング』で 22 個紹介されている

不吉な匂いをしめすセリフ

- ソースコードを見ていて次のようなセリフを口にしたくなったらリファクタリングが必要である可能性がある
 - 「ダブってる!」: 重複したコードが多い
 - 「長すぎ!」: メソッドが長過ぎる, クラスが大きすぎる
 - 「多すぎ!」: クラスが多すぎる
 - 「名前が合っていない!」: 変数名などが適切でない
 - 「公開しすぎ!」: メソッドやフィールドが `public` であるものが多い
 - 「オブジェクト指向っぽくない!」: オブジェクト指向の特性を利用していない

リファクタリング・カタログ

リファクタリング・カタログとは

- リファクタリングの目的や手順をカタログ化してまとめたもの
- 書籍『リファクタリング』や以下のウェブサイトでまとめられている
 - <http://www.refactoring.com/>

システマティックな修正

- カタログに従ってシステマティックにコードを変更する
- 退屈に思うこともあるが、ミスを少なくすることが出来る
- カタログはこれまでの多くの熟練した設計者やプログラマの経験を整理したもの

リファクタリングのエッセンス

ステップ・バイ・ステップ

- 1 度に 1 つずつ行うことがリファクタリングの原則
- 2 つの修正を 1 度に行わない
 - 同時に修正している間に何の修正をしているかわからなくなり, ミスを招く
- 後戻りしやすいように
 - ある手順で間違いがあった場合, 間違いをしていない手順まで戻ることが出来る

ステップ・バイ・ステップ

- ステップごとの確認
 - ある手順を終える度にコンパイルやテストを行う
- 古いものを新しい物に変えていく
 - 「すべてを壊して, すべてを作り直す」のではなく「動く状態を保ちつつ新しいコードを追加していき, 古いものが全て新しくなったところで, 古いものを壊す」
 - こうすることでミスを減らすことが出来る

リファクタリング Q&A

- リファクタリングは万能薬か
 - 全ての問題を解決するものではなく, バグを取りやすく, 機能を追加しやすいようなコードに改善するだけ
- リファクタリングを学ぶ価値はあるか
 - 経験を詰んだプログラマは自分の経験の再整理・再確認になる
 - 初心者プログラマは先人の知恵を効率よく身につけることが出来る
- 動いているコードに手を入れていいのか
 - 動いているコードに手を入れることはバグを発生させる原因になる
 - リファクタリングを注意深く行い, 読みやすいコードを作る必要がある

- どこまでが「外部」なのか
 - メソッドをリファクタリングする場合、メソッド本体が「内部」でメソッドを呼び出す側が「外部」となる
 - テストによって「外部」にリファクタリングの影響が及ばないことを確かめる必要がある
- 初期設計をしっかりとるほうが有効ではないか
 - 設計段階で見つけられなかった問題を実装段階で見つけることが多い
 - ソフトウェアに対する要求の変化は頻繁に変化する
 - 設計段階で見つけられなかった問題を解決・軽減するためにリファクタリングを行う
 - そのためリファクタリングは「設計の改善」とみなすことも出来る

- 適切なリファクタリングを見つけるには
 - 「不吉な匂い」を嗅ぎつけられるようになる
- 全部覚える必要はあるか
 - いきなりそうする必要はなく, 使っていく中で覚えれば良い
- エクストリーム・プログラミングと関係があるか
 - エクストリーム・プログラミングはアジャイル型開発の一種
 - 「現実にあわせて, 少しずつ設計が進んでいく」という性質に関連している

Q&A

- デザインパターンと関係があるか
 - リファクタリングが向かう「よりよいコードの在り方」がデザインパターンと考えて良い
- 今の職場でリファクタリングは難しい
 - 予算と時間を考えると導入するのは怖い
 - 丁寧にリファクタリングを行えばバグの除去や機能追加がしやすくなるため将来的に予算と時間を節約することが出来るかもしれない

まとめ・所感

まとめ

- リファクタリングによって「コードの改善」を行うことでバグの除去や機能追加しやすくなる
- 「不吉な匂い」を嗅ぎつけてリファクタリングを行う
- リファクタリング・カタログに従ってリファクタリングを行い, ステップ・バイ・ステップでミスを減らす

所感

- リファクタリングは諸刃の剣だと思った
 - ミスする可能性があり, 新たにバグを入れてより面倒なことになる可能性がある
 - リファクタリングのリスクを軽減するための研究を調べてみたいと思った
- 素人でも学ぶ価値はあることを知れてよかった
 - リファクタリングから先人の知恵を学んで, コードを書く際に活かしたいと思った
- 「不吉な匂い」を嗅ぎつけるにはある程度の能力がいるんじゃないかと思った
 - おそらく「不吉な匂い」を自動的に検出する研究やツールがあると思うので調べてみたいと思った