

# オブジェクト指向エクササイズのス スメ

(このページはあとテンプレートに  
替えます)




ところで  
オブジェクト指向開発  
していますか？



# 本当にオブジェクト指向?

- 言語とフレームワークはオブジェクト指向
- データと処理は分離する
- 開発者は用意されたクラスを実装するのみ
- ポリモーフィズムなどを使わない

オブジェクト指向でない事自体は問題ではないが。。



# できない理由


- ふつうの開発者には理解できない
- 一部のマニアックな開発者のおもちゃ
- そもそもオブジェクト指向自体にメリットがあるかどうかがわからない

ネックは教育



# そこで オブジェクト指向エクササイズ

オブジェクト指向開発を強制的に身に着けるため  
にハードなコーディング規約をプロジェクトに  
適用するエクササイズ



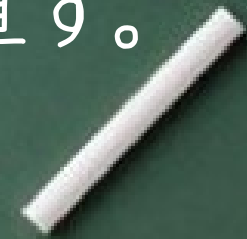
しばし、オブジェクト指向エクササイズの  
内容説明を・・・



# 実際に試してみる

## 「お題」

一見00っぽい感じだけど、エクササイズの観点で見るとダメダメな、あるツールをエクササイズのルールに従うように書き直す。





# Amazon中古価格調査ツール

単純なコマンドラインツールを

リファクタリングします

- 引数でASINを与えて起動
- AmazonマーケットプレイスをWebAPIで呼び出し、最も安い値段を検索する
- 標準出力に出力する





# 当初の構造



# 視点

- XXXX次のページにつなげるためのフック  
が必要



# クラスの抽出に役立つルール

- すべてのエンティティを小さくすること
- すべてのプリミティブ型と文字列型をラップすること
- 一つのクラスにつきインスタンスは二つまでにすること?



- XXXXX次のページにつなぐためのフックが必要



# カプセル化に関するルール

- getter/setterを利用しない



つらい・・・

setter排除はなんとか出来るが、getter排除は非常に困難!!!

例えばこんな時は、Bookから情報取得しないとどうしようも無い

- 画面表示
- バリデーション



# 考えられる戦略

- 妥協する(エクササイズ的にはNG)
- プラガブルMVC的発想
- Visitor的発想
- etc . .





# プラグブルMVC的発想

利用者(この例では画面)が要求するインタフェースをモデル側(この例ではBook)に実装する



コード例



# 利点と欠点



# Visitorパターンを利用する

画面表示用のVisitorオブジェクトをBookに占有させる



コード例



# 利点と欠点



# その他

- Bookクラス自体に表示能力を持たせる



どの案がよいか

月並みな発言だが正解は無い



# 値オブジェクト戦略

この戦略では、画面表示にModelの内部の実装が  
引きずられる



# Visitor戦略

そもそもGUIで適用できるかどうか微妙

しかし画面表示ではなく、Modelオブジェクトの  
バリデーションなどには便利かもしれない



# 結局

- エクササイズが議論を引き出した



まとめ



# オブジェクトHogeを巡る進化

- オブジェクト指向言語
- オブジェクト指向設計
- オブジェクト指向分析設計
- オブジェクト指向開発プロセス
- 開発プロセス
- . . .



# オブジェクト指向設計

オブジェクト指向言語だけでは、望んでいたパワーを得ることができない!

- 設計という言葉が注目されたのは1990年代後半
  - デザインパターン(1999)
  - UML1.0(1999)

日本でもメーリングリストやNewsgroupでも、昼夜、さまざま議論が行われていた



# オブジェクト分析設計への進化

オブジェクト指向設計は、開発のライフサイクル全体を、より初期の工程(嫌な言い方をすれば上流工程)を含むように進化していった



# オブジェクト指向開発プロセス

オブジェクト分析設計は、開発における様々な活動(構成管理や各種マネジメントを含む)を含む開発プロセスに進化していった



しまいには、

開発プロセスは、個々の技術要素に対して非依存  
になっていった。



# なぜならば・・・

魔性のワード「属人性の排除」

- 開発プロセスは、誰がやっても同じようにできて、すぐに適用出来るべき
- 設計という行為は個人のスキルに依存するので、開発プロセスの話題とは分離したい



# FWにおける属人性の排除

フレームワーク開発者はオブジェクト指向を駆使してすごいのを作り上げるので、利用者はテンプレートを埋めていくだけでよいですよ!!



# 設計を軽視

- 重量級のプロセスでは
  - 最も優秀な人が設計を行い、プログラマがそれを展開するのだ
- アジャイルプロセスでは
  - アジャイルな感じでプロセスを回していくと自然によい設計に近づいてくのだ



そんなわけじゃない





# いつでも設計

- 設計とは、プログラムを記述する際に、判断の根拠をあたえるもの
- 判断はミクロからマクロまであらゆる局面で必要になり、その場で判断基準を決定する事も多い。



# みんなで設計

- 設計時の判断の集積はアーキテクチャになり、コンセンサスになる
- 自分と他の開発者、未来の開発者をつなぐソーシャルな行為である
- だから、みんなが設計できなければ意味が無い



# スキルを軽視した開発の失敗

そんな設計行為には、個々の開発者のスキルが重要で、

アジャイル開発プロセスでも設計を考慮しなければ失敗する

- <http://www.infoq.com/jp/news/2008/11/decline-of-agile>

「人は大地から離れては生きていけない」

リュシータ・トエル・ウル・ラピュタ(1986)



# 閑話休題

デザインパターンなどのパターン言語は、頭を使わないで設計するためのノウハウ集じゃないの？ 属人性を排除するものではないの？

- 違う、それは人をより能動的に設計プロセスに参加させるための道具です！



# エクササイズのリット

- ある程度現実的な課題でできる
- 元になる課題が簡単であっても、エクササイズのルールを適用する負荷が高い
- 各種、デザインパターンなどのテクニック、知識を引き出す必要がある。
- 多人数で開発している場合は、議論を巻き起こす

誰がやる？





