単体テストの観点から見たデザイン・パターン

日本アイ・ビー・エム株式会社
技術　ソフトウェア・エンジニアリング
太田健一郎

アジェンダ

- 実は単体テストが原因？
- 効果的な単体テストって？
- 設計から考えないと駄目だよね
- テスト容易性とデザイン・パターン
- テストの観点から見てみよう
  - 基本パターン
  - エンタープライズ・システムのパターン
- まとめ

よくあるシステム開発後半の修羅場

- システムテストのフェーズで欠陥が多発する

何で，こんなにバグがたくさん出るんだ！！
本当に単体テストをやったのか？
そもそも，こんなバグは単体テストで発見できる
だろう！！
こんなバグを出すのは自分の仕事じゃない！！

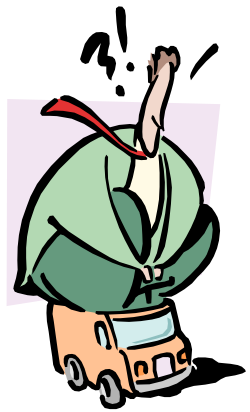
テスト担当者

それなりにテストはしたよ。
何で開発していないテスト担当者にそこま
で言われる筋合いがあるんだ！！
デバッグで手一杯なんだか勘弁してくれよ。
こっちも寝てないんだ！！

開発者

原因のひとつ，それは単体テスト

- テスト計画や網羅基準も存在するにもかかわらず
- 実は
  - 本来，開発者が単体テストで発見できる欠陥を摘出し切れていない
    - あんまりテストのことは考えないで開発している
    - 単体テストを実行するのが難しいプログラムなっている
    - 一応，網羅基準は満たすようなテストはするが時間と手間がかかる
    - 要するに，単体テストがうまく回っていない
    - 当然，欠陥が発生した場合の問題の切り分けも難しくなる



効果的な単体テストって？

- 一度に一つのことを小さな要素から積み重ねてテストできる
- 依存関係を最小限にして要素をテストできる
- より小さな要素で網羅率の高いテストが実行できる
- 各要素のテストは複雑な手順を踏むことなく，繰り返して実行できる

設計から考えないと駄目だよね

効果的な単体テストを実施するためには，設計段階からテスト容易性(テストのしやすさ)を考慮する必要がある

- テストがしやすいクラスの設計
  - 要素の依存関係が最小限になっている
  - 要素(メソッド，クラス，クラスの集合)が依存する別の要素をダミーの要素に置き換えられる
  - ダミーの要素は最小限の手間で作成，設定できる

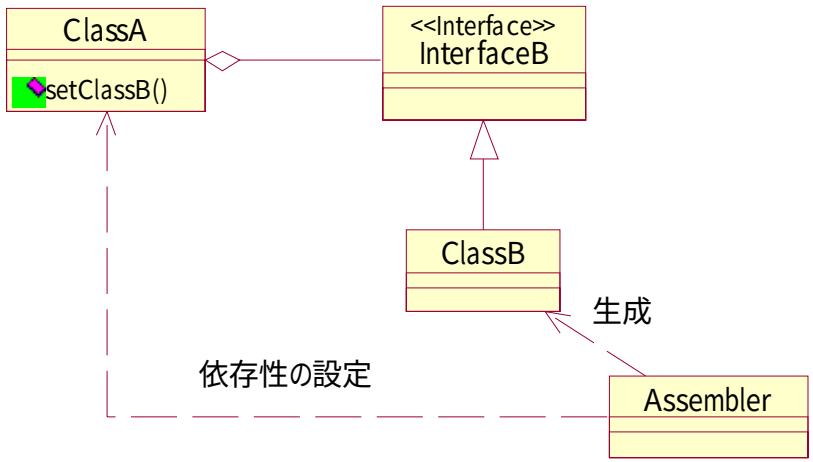
テスト容易性とデザイン・パターン

- デザイン・パターン
  - 開発者が設計の指標として利用する
  - 状況ごとに発生する問題を解決するためにトレードオフを考慮したうえで利用する
- テスト容易性とデザイン・パターン
  - 設計の考慮点としてテスト容易性を考える場合，デザイン・パターンをテストの観点から見てみるのは有用ではないか
  - 本来の意図とは異なるが，結果としてテスト容易性に役立つようなデザイン・パターンが存在するのではないか

基本パターン

- Dependency Injection
- Mock Object
- Don't talk to strangers, The Law of Demeter

Dependency Injection

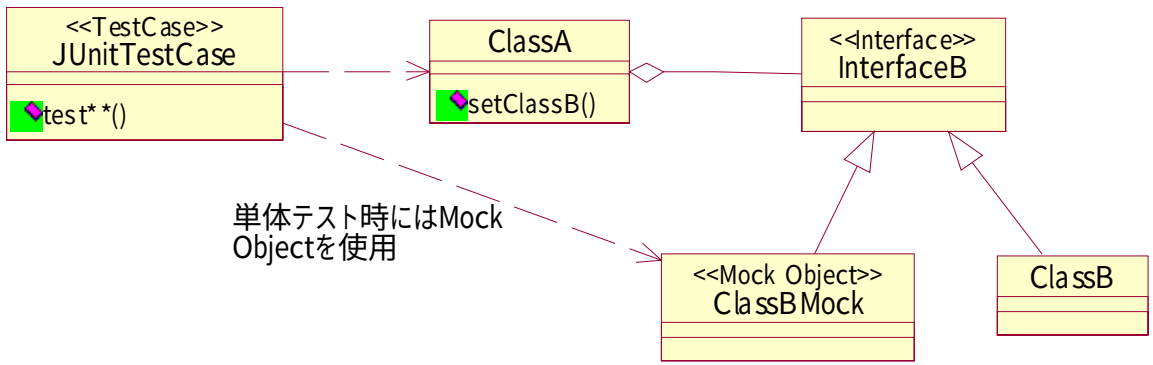


- **概要**
  - **独立したオブジェクトをAssembler(組立て係)として用意し，インターフェースBの実装(オブジェクトB)をオブジェクトAの属性に設定させる**
  - **オブジェクトの生成と利用を分離し，依存するオブジェクトをコンパイル時ではなく，生成時にリンクできるようになる**

- **テストの観点からの考察**
  - **オブジェクトBをダミーのオブジェクトに簡単に置き換えられるため，オブジェクトAを依存するオブジェクトBから切り離して単独でテストできるようになる**

参考:
Inversion of Control **コンテナと** Dependency Injection **パターン**
http://www.kakutani.com/trans/fowler/injection.html

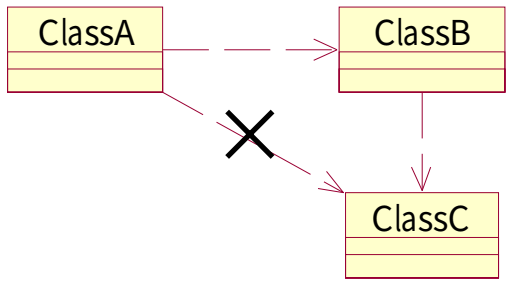
Mock Object



- **概要**
  - **テストの対象となるクラスAが依存，協調動作するオブジェクトBと同様のインターフェースを持ち，振る舞いを偽装するオブジェクト**
  - **テスト対象となるオブジェクトAは依存するオブジェクトBのインターフェースにのみ依存しているため，オブジェクトBが本物か偽者かであることを意識することはない**
  - **Mock Objectは単なるスタブとは異なり，メソッドが呼び出された順番，回数等を自己検証する機能を持ったオブジェクトである**

- **テストの観点からの考察**
  - **ダミーのオブジェクトとなるMock Objectを利用することによって、オブジェクトAをテストする際に依存する実オブジェクトBを利用せずに済むため，以下のような効果が期待できる**
    - **テスト対象のクラス単独でテストが可能になる**
    - **テストの際に，一般に複雑になる依存オブジェクトの設定が不要になる**
    - **例外(データベースのエラー，ネットワークのエラー等)を返すMock Objectを用意することにより，実オブジェクトでは発生させるのが難しい例外系のテストを容易に実施できる**
    - **実オブジェクトの初期化／終了処理を行う必要がないため，テストの実施時間を短縮できる**
    - **インターフェースが決定していれば，依存するオブジェクトの完成を待たずにテストを実施できる**
    - **Mock Objectの自己検証機能を使って，依存オブジェクトを正しく利用しているかを検証できる**

Don't talk to strangers
The Law of Demeter



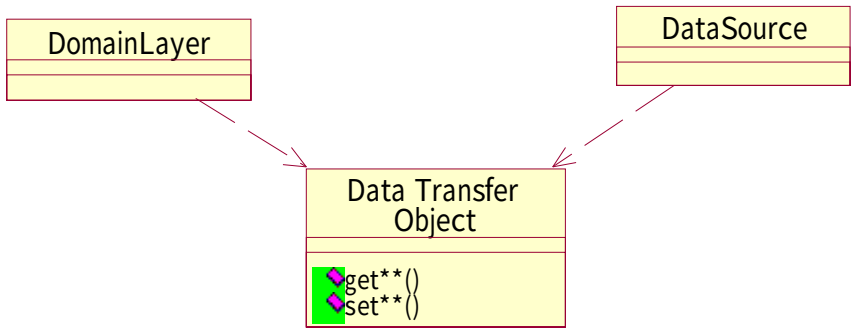
- 概要
  - あるメソッドの中では直接的に依存するオブジェクトに対してしか通信を行わない
    - 直接的に依存するオブジェクト
      - オブジェクト自身
      - オブジェクトの属性
      - パラメータとして渡されたオブジェクト
      - メソッド内で直接生成したオブジェクト

- テストの観点からの考察
  - テストの際に，Mock Objectによる置き換えの対象となるオブジェクトが間接オブジェクトをMock Objectとして返す必要がなくなるため，Mock Objectを作成する際の手間と検証を簡易化できる
  - テスト対象のオブジェクトAが直接依存するオブジェクトBが，内部で外部環境に依存する間接オブジェクトC(コンテナ，外部リソースなど)を利用している場合でも，Mock Objectを作成する際にその間接オブジェクトCを準備する必要がなくなる

エンタープライズ・システムにおけるパターン

- Data Transfer Object
- Data Mapper
- Template View

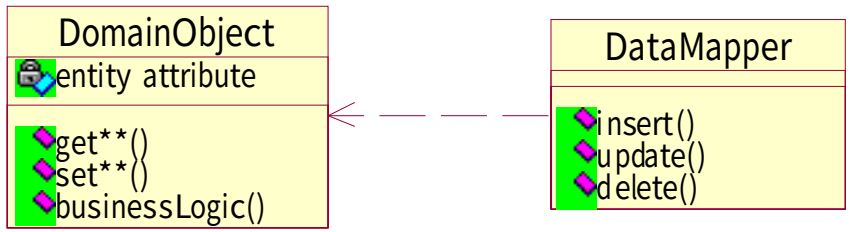
Data Transfer Object



- **概要**
  - **レイヤーごとに異なった物理リソースを割り当てる場合を考慮し，各レイヤー間でのメソッド呼出しの回数を減らしRPCのパフォーマンスを向上させるために，データ転送専用のオブジェクト(Data Transfer Object)を用意し，レイヤー間の通信にはこのオブジェクトを利用する**
  - **このData Transfer Objectはデータの設定･取得以外の振る舞いを持たず，ネットワーク上でシリアライズ可能にするために属性値として単純なプリミティブ型もしくはシリアライズ可能なオブジェクトを使用する**

- **テストの観点からの考察**
  - **各レイヤー間の通信にデータを保持するだけの単純なオブジェクトを用いることによって，レイヤー間の関係を疎にできるため，レイヤーを分離したテストが容易になる**
  - **Data Transfer Objectはデータの設定･取得以外の複雑なロジックを持たないため，振る舞いを偽装するMock Objectを用意することなく，そのままData Transfer Objectをテストに利用できる**

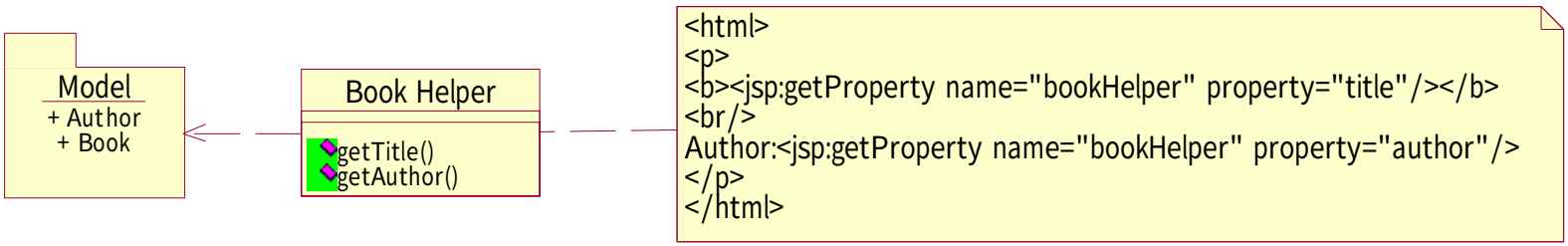
Data Mapper



- **概要**
  - 現在のエンタープライズ・システムの永続化にはリレーショナル・データベースを用いる場合が多いが，オブジェクトとリレーショナル・データベースにはデータ構造と振る舞いにインピーダンス・マッチが存在する
  - ビジネスロジックもしくはデータベースの操作が複雑な場合，ビジネスロジックを実行するオブジェクト内で直接，データベースの操作を記述すると，オブジェクトの構造と振る舞いがデータベースの構造に大きく依存する，コードの構造が分かりにくくなる，トランザクションの管理が難しくなる等の不具合が出てくる
  - そこで，ビジネスロジックを実行するオブジェクト内で直接，データベースの操作をする(SQLの発行等)と，オブジェクトとデータベースをマッピングするオブジェクト(Data Mapper)を設けて両者を分離する

- **テストの観点からの考察**
  - Data MapperにMock Objectを使用することにより，実際のデータベースを用意することなく，データベースから独立してビジネス・ロジックをテストできる
  - Data Mapperとの通信にData Transfer Objectを用いることによって，ビジネス・ロジックから独立して，データソース・レイヤー(Data Mapper自身)をテストできる

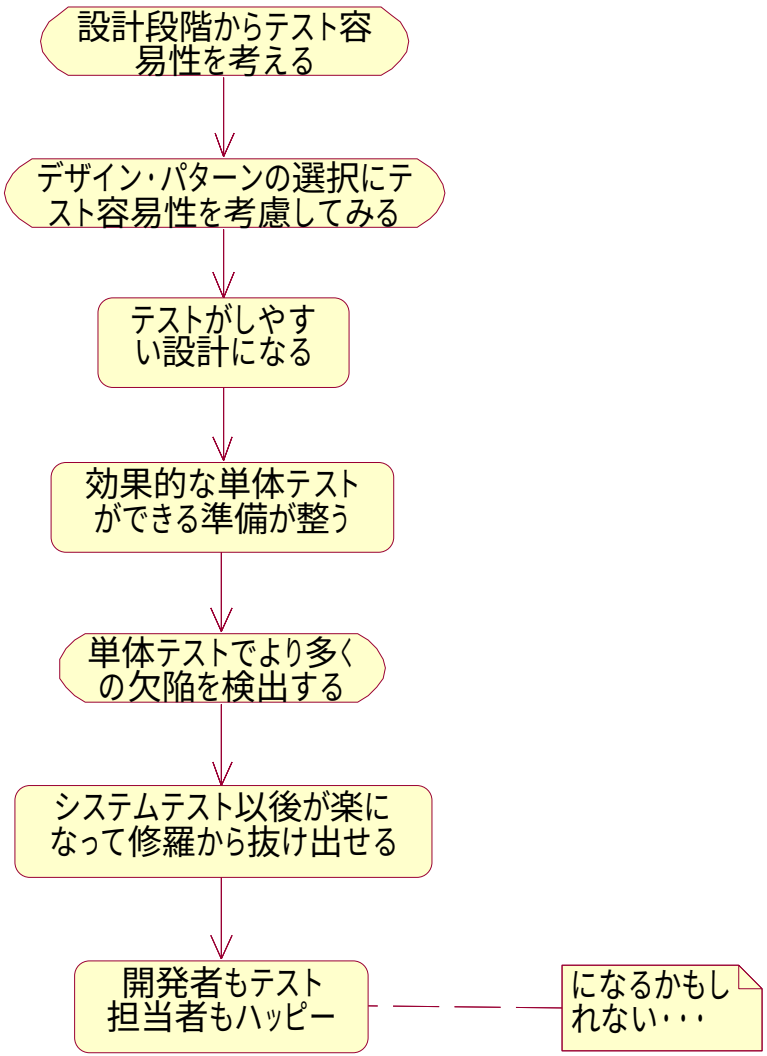
Template View



- **概要**
  - **プレゼンテーション･レイヤーのオブジェクト (JSP等) が必要とする情報を保持するHelperオブジェクトを用意し，プレゼンテーション･レイヤーのオブジェクトが直接ドメイン･レイヤーのオブジェクトに依存しないようにする**
  - **Helperオブジェクトはドメイン･レイヤーのオブジェクトから情報を取得し，プレゼンテーション･レイヤーのオブジェクトが利用しやすい形に変換する**
  - **プレゼンテーション･レイヤーのオブジェクトは表示用に画面をレイアウトする部分のみを担当する**
  - **プレゼンテーション･レイヤーからビジネス･ロジックに依存するコードを除くことができる**
  - **プレゼンテーション･レイヤーを担当する開発者(通常はデザイナー)はシステムのロジックをほとんど意識することなく，従来の静的コンテンツを作成するのを同じように開発できるようになる**

- **テストの観点からの考察**
  - **プレゼンテーション･レイヤーの入出力のテストを実行する際に，Helperオブジェクトをテスト結果の検証に用いることによって，最終的に整形される画面に対するパースを実行することなく出力する値の検証ができる**
  - **HelperオブジェクトにMock Objectを利用することにより，JSPのようなプレゼンテーション･レイヤーのオブジェクトを単独でテストできるようになる**

まとめ


設計段階からテスト容易性を考える
デザイン・パターンの選択にテスト容易性を考慮してみる
テストがしやすい設計になる
効果的な単体テストができる準備が整う
単体テストでより多くの欠陥を検出する
システムテスト以後が楽になって修羅から抜け出せる
開発者もテスト担当者もハッピー
になるかもしれない・・・

謝辞

- 本資料をレビュー頂いたテストパターンサブタスクのメンバーに感謝いたします
  - 小井土さん
  - 福井さん
  - 山野さん
  - 山下さん
  - 小坂さん

ディスカッション

- 参加者の皆様がテストのために工夫されていること
  - 要件定義
  - 分析
  - 設計
  - 構成管理
  - パターン
  - ツール
  - 人的側面