TDDを加速する擬似クラス生成方法

日本IBM
伊尾木 将之，河野 広伸，合田 英二

まずは結論から

- SCT(Simulated-Class by Tests)という手法のお話
  - テストから擬似クラスを生成する手法
  - TDDと親和性が非常に高い

以上

ご質問は？

TDDとは

- TDD（テスト駆動型開発）とは
  - テストを用いて設計・実装を進化させていく手法
  - テストファーストとリファクタリングを融合した手法としてKent Beck氏が提唱
  - アジャイル手法の中では比較的普及している
  - テスト容易性を重視

TDDとは

- TDD（テスト駆動型開発）とは
  - テストを用いて設計・実装を進化させていく手法
  - テスト
  - アジャ
  - テスト容

eck氏が提唱

省略

擬似クラスとは

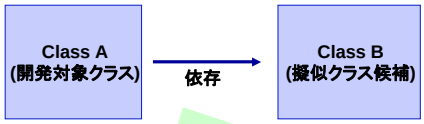
- いわゆるモック・スタブ・ドライバなどのこと

擬似クラス:
ロジックは正しくないが、外からは
正しいクラスと同じ振る舞いを見せる擬似のクラス

- 最近擬似クラスを使用した開発が注目されている
- TDDでも活用されている

擬似クラスが使用される例

- 以下のような状況でよく使用される
  - 未実装のクラスに依存するクラスの開発
  - 環境に依存するクラス
    - …. etc



・Class Bが未実装
・環境などに依存
・他チームが開発中　等

このような状況では本当に開発したいClass Aが
実装・テストが出来ない
→擬似クラスでClass Bを代用

擬似クラス生成の問題

- 擬似クラスの生成問題
  - 擬似クラスは有用であるが、擬似クラスを生成するのは容易ではないのでツールを使用する。しかしそれでも容易ではない
  - 擬似クラスの振る舞いを定義方法が難しい
    - 入力値の定義、出力値の定義、メソッドの呼び出し順の定義など
  - 擬似クラスの振る舞いの定義が生成ツール独自のAPIに依存する
    - 学習コストが高くなる
  - 擬似クラスの振る舞いの定義を苦労して作成しても本物クラスが作成されると破棄しなくてはいけない
    - もったいない

擬似クラスの定義の一例

- 某モック生成ツールの擬似クラスの定義のサンプル

```
import org.jmock.*;
class PublisherTest extends MockObjectTestCase {
  public void testOneSubscriberReceivesAMessage() {
    // set up
    Mock mockSubscriber = mock(Subscriber.class);
    Publisher publisher = new Publisher();
    publisher.add((Subscriber) mockSubscriber.proxy());
    final String message = "message";

    // expectations
    mockSubscriber.expects(once()).method("receive").with( eq(message) );

    // execute
    publisher.publish(message);
  }
}
```

この定義って分かりやすいですか？
苦労してAPI理解して書いたこのコードって
捨てるのもったいなくないですか？

ところでJUnit形式のテストを考える

- JUnit形式のテストは・・・

```
import junit.framework.TestCase;
public class CalcTest extends TestCase {
    Calc calc;
    protected void setUp() throws Exception {
        super.setUp();
        calc=new Calc();
    }

    public void testAdd(){
        calc.set("+");
        assertEquals( 5, calc.get( 2, 3));
    }
}
```

入力・出力・メソッドの呼び出し順など
非常に分かりやすくないですか？

ところでJUnit形式のテストを考える

- JUnit形式のテストは・・・

```
import junit.framework.TestCase;
public class CalcTest extends TestCase {
    Calc calc;
    protected void setUp() throws Exception {
        super.setUp();
        calc=new Calc();
    }

    public void testAdd(){
        calc.set("+");
        assertEquals( 5, calc.get( 2, 3));
    }
}
```

これって擬似クラスの定義に使えそう？

そこで!!

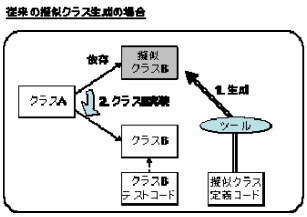
SCT!

つまり

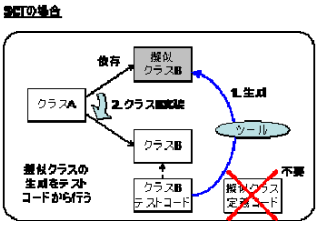
擬似クラスの定義をテストで代用しましょう！

SCTとは

- テストを擬似クラスの定義と捕らえ、テストから擬似クラスを生成する手法



従来の方法



SCTの方法

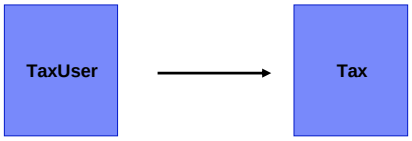
SCTの利点

- テストを擬似クラスの定義と捕らえる
  - テストコードは分かりやすい
  - 学習コストが低い
  - JUnitはデファクトスタンダード→さらに学習コストは低い
- テストコードは廃棄しない
  - 擬似から本物のクラスに変わってもテストコードは使用可能
- TDDと親和性が高い
- Javaでの実装：機械猫モッカー
  - オープンソース

と、いうわけで

デモ

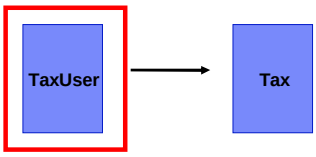
デモ概要



Tax：消費税を計算するクラス
TaxUser：Taxクラスを使って、消費税を表示するクラス

Taxクラスは未実装 → 擬似クラスで代用

デモ概要

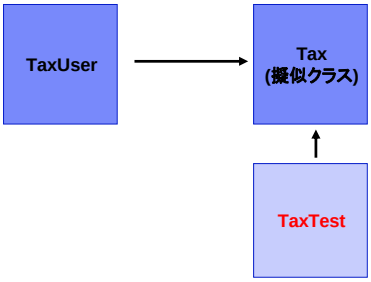



```
class TaxUser{
  public static void main(String[] args){
      Tax tax=new Tax();
       System.out.println("100円は" + tax.calc(100) + "円になります");
  }
}
```

こんな風にTaxクラスを使用したい。

Taxクラスがまだなーい

デモ手順



1.Taxクラスの擬似クラスを生成するためにTaxTestを書く
2.TaxTestを機械猫モッカーに渡す
3.おわり

JavaでのSCT実装である機械猫モッカーを使用

SCTでの可能性

- 思考錯誤の容易化
  - テストを作ってためす
- テストでのコミュニケーションが図れる
  - テストがあればすぐ動かせる
- シーケンス図などの設計の確認が容易になる
  - 分析・設計段階でも簡単なテストはかける

- TDDとの関連
  - TDDはテスト駆動・SCTはテストを利用する→どちらもテストを中心にした考え方
  - TDDwithSCTで、テストの新しい利用方法を提案する