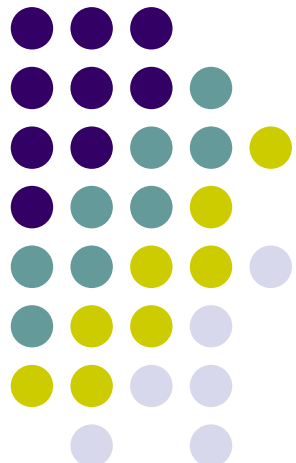
ワークショップ
テスト駆動開発テクニック＆
最新アジャイルテスティング事情



クリスマス会

オブジェクト倶楽部　北野弘治

アジェンダ

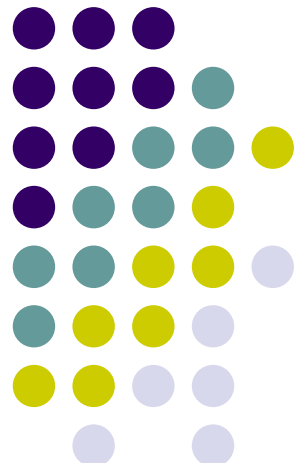
- 講師紹介 5分
- JUnitの使い方 5分
- テスト駆動開発の基本 20分
- テスト駆動開発を体験しよう 20分
- 最新テスティングツールFITの紹介 20分
- FITのデモ 10分
- FIT課題 制限時間まで
- 質疑応答 適宜

講師紹介：北野弘治

- 株式会社永和システムマネジメント所属
  ITコンサルタント
- キーワード：
  OO、Web、XMLなど
- アジャイル活動：
  ADC2003参加
  アジャイルプロセス協議会
  見積契約WGリーダ

外人とペアプロ。ペアプロは言語を超えた。

JUnitの使い方



JUnitの設定

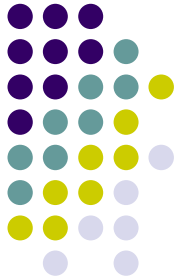
- ダウンロードします。現在、3.8.1が最新です
  - 入手先URL: http://www.junit.org/index.htm
  - junit.3.8.1.zipをダウンロードします
- ダウンロードしたファイルを展開します
- 展開したファイルのうち、junit.jarをc:¥junitにコピーします
  - コピーしなくてもよいが、説明を簡単にするためコピーしています。
- junit.jarにコンパイルおよび、実行時にパスを通します
  - javac -classpath c:¥junit¥junit.jar JUnitTest.java
  - java -cp .;c:¥junit¥junit.jar junit.textui.TestRunner JUnitTest

テストケースの作り方

- junit.framework.TestCase を継承したクラスを作成する。
- public void testXXXX() メソッドを作成する。

```
import junit.framework.TestCase;

public class JUnitSample extends TestCase{
    public void testSample1(){
        System.out.println("テストケースが実行されました");
    }
}
```

テストケースの実行



- クラスパスに、実行するテストケースクラスのパスとJUnitのライブラリを含ませて、テストランナーを実行する。引数にはテストケースクラス名を指定する。

```
C:\sample>java -cp .;c:\junit\junit.jar junit.textui.TestRunner JUnitSample
.テストケースが実行されました

Time: 0

OK (1 test)
```

代表的なassertメソッドの使い方

- assertTrue
  - **引数に、真となる式を与える**

```
assertTrue(Math.abs(-1) >= 0);
```

- assertEquals
  - **引数に、期待値と期待値になるであろう式を与える**

```
assertEquals(3, 1 + 2);
assertEquals(“abcd”, “ab” + “cd”);
```

- fail
  - **失敗メッセージを出力する**

```
fail(“失敗”);
```

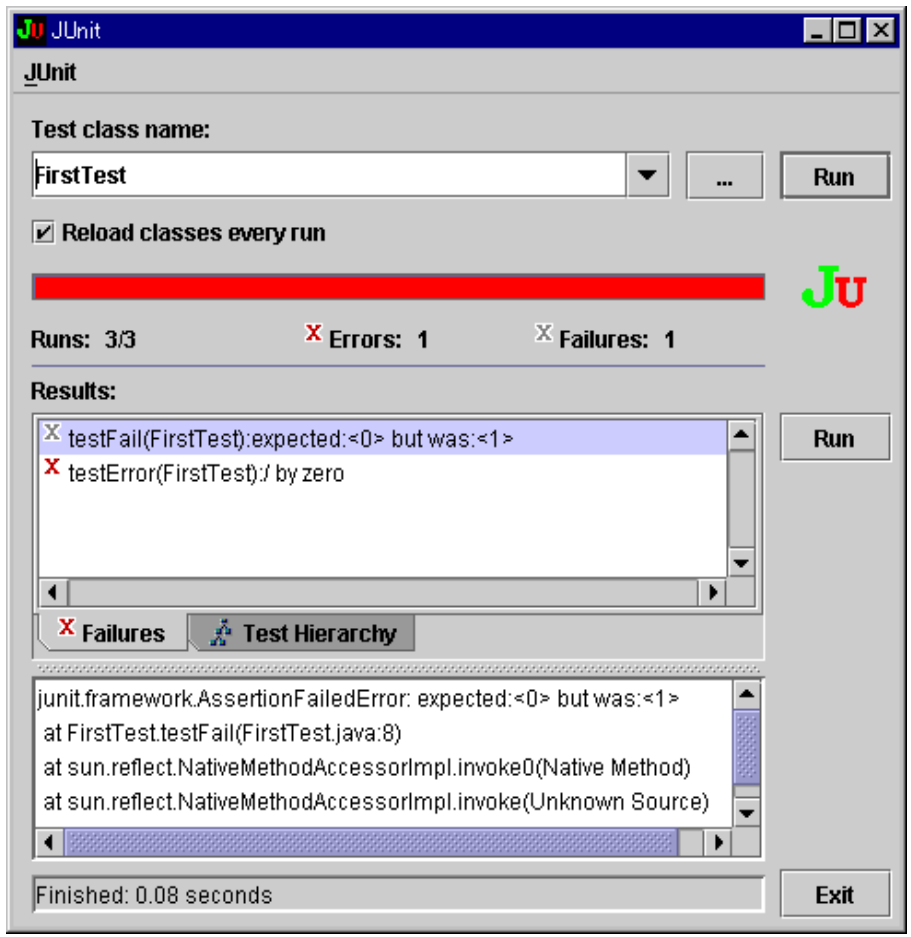
以下のプログラムを作成し、実行してください

c:¥angya¥junit¥FirstTest.java

```
import junit.framework.TestCase;

public class FirstTest extends TestCase{
    public void testSuccess(){
        //true？とtrueに聞いてるのでテストは成功
        assertTrue(true);
    }
    public void testFail(){
        //0を期待しているが1としているのでテストは失敗
        assertEquals(0, 1);
    }
    public void testError(){
        //ゼロ割エラーが発生する。
        int i = 0/0;
    }
}
```

```
c:¥angya¥junit>javac -classpath c:¥junit¥junit.jar FirstTest.java
c:¥angya¥junit>java -cp .;c:¥junit¥junit.jar junit.swingui.TestRunner FirstTest
```

実行結果


JUnit
JUnit
Test class name:
FirstTest
...
Run
Reload classes every run
Runs: 3/3
Errors: 1
Failures: 1
Results:
testFail(FirstTest):expected:<0> but was:<1>
testError(FirstTest):/ by zero
Run
Failures
Test Hierarchy
junit.framework.AssertionFailedError: expected:<0> but was:<1>
at FirstTest.testFail(FirstTest.java:8)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
Finished: 0.08 seconds
Exit

プロダクトクラスとテストケースクラス

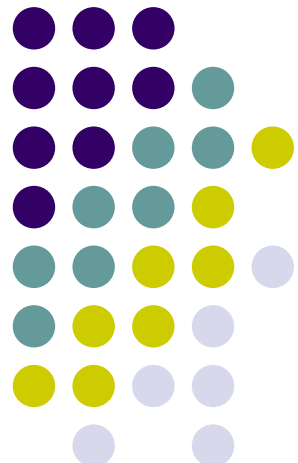
```
public class Calc{
   public int add(int a, int b){
        return a + b;
   }
}
```

プロダクトクラス

```
import junit.framework.TestCase;

public class CalcTest extends TestCase{
   public void testAdd(){
      Calc c = new Calc();
      assertEquals(3, c.add(1, 2));
   }
}
```

テストケースクラス

テスト駆動開発の基本



テスト駆動開発
(TDD:Test Driven Development)

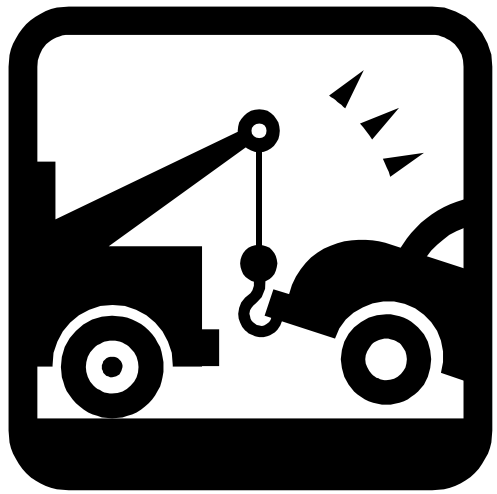
- 以前、テストファーストと呼ばれていた作業の進め方・考え方を、より体系立て、プラクティスにまで昇格させた。
- テストを作りながら、設計を考える。
- コードカバレッジC1レベル（命令網羅）は達成する

TDD(テストファースト) 効果

- 最初にテストを書くことにより、仕様把握をしているのと同じ効果が得られる(仕様を知らなければ組めないため、実装において仕様把握は重要であり、テストファーストはそれを強制している)
  - 最初に目的地をきっちり決める。
- 目標を最初に決め、そこに向かってコーディングするため、無駄なコードが無くなる
  - 着実に目的地の方向に進む。
- 開発のスピード調整が可能
  - 危険な交差点はゆっくり確認しながら。。。
  - 高速道路はかっ飛ばせーーー
- テストしやすい設計になる
  - 標識が沢山ある。

テストファーストをしていない場合

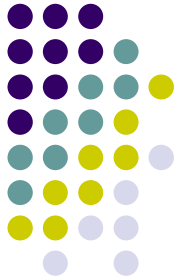
- 実装したコードの正当性を確認するテストをしていませんか？？
  - テストするのは、仕様が正しく実装されているか？です。
- 無駄に書いたコードのテストまでしていませんか？？
- テストしにくい（他の人から使いにくい）設計になっていませんか？
- 実装している途中に、なにか他のことをやっていませんか？



TDD十訓

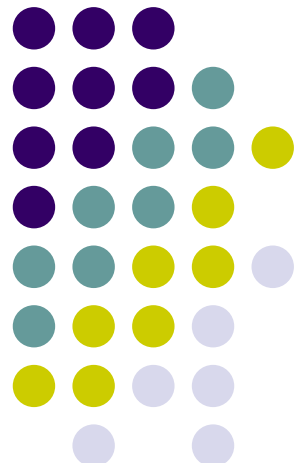
- ToDoリストを書くべし
- 最適ペースでテスト・実装するべし
- テストを書いたら、まず『赤』になることを確認するべし
- 『赤』→『緑』→『赤』→『緑』・・・のテンポを守るべし
- 答えが思いつかない場合は、とりあえず相手を騙すべし
- 『緑』になったからと言って安心するべからず
- 常に不吉な臭いを感じる鼻を持つべし
- コンパイラを信じるべし
- まずは1つのテストから。時には2つのテストで責めてみるべし
- 自分を信じるべし

TDDの手順



1. テストを書く
2. コンパイルが通る最小のコードを記述する
3. テストに失敗することを確認する　(赤)
4. テストが通る最小のコードを記述する
5. テストが通ることを確認する　(緑)
6. 不吉な臭いがしたら、勇気を持ってリファクタリングする
   (1) リファクタリングが成功し、テストが通ることを確認する　(緑)
7. 仕様が満たされるまで1〜6を繰り返す

テスト駆動開発を体験しましょう



TDDによる四則演算クラスの作成(1/13)

- **戦略を練る**
  - **クラス名は**「Calc」
  - **整数**(int)**の四則演算のみ行う。長整数、実数は対応しない**
  - **メソッド名は**「add」「sub」「mul」「div」**とする**
  - **まずは、足し算を実装する。**2**つの整数を引数とする**

ToDo

足し算
引き算
掛け算
割り算

TDDによる四則演算クラスの作成(2/13)

- テストケースクラスを作成する
  - 「Calc」クラスのテストケースクラスなので「CalcTest」クラスとする。

c:¥angya¥calc¥CalcTest.java

```
import junit.framework.TestCase;

public class CalcTest extends TestCase{
}
```

TDDによる四則演算クラスの作成(3/13)

- テストメソッドを追加する
- 足し算「add」をテストするメソッドなので「testAdd」という名前にする
- テストメソッドのおまじないである「public void」も忘れずにつける

c:¥angya¥calc¥CalcTest.java

```
import junit.framework.TestCase;

public class CalcTest extends TestCase{
    public void testAdd(){
    }
}
```

TDDによる四則演算クラスの作成(4/13)

- テストケースクラスをコンパイルする。

```
c:\angya\calc>javac -classpath .;c:\junit\junit.jar CalcTest.java
```

- テストケースを実行する。
  - 別のプロセスでSwingベースのテストランナーを起動する

```
c:\angya\calc>java -cp .;c:\junit\junit.jar junit.swingui.TestRunner CalcTest
```

TDDによる四則演算クラスの作成(5/13)

- テスト表明を記述する
  - 1+2=3を確認する
  - 「add」メソッドはインスタンスメソッドとする

c:¥angya¥calc¥CalcTest.java

```
import junit.framework.TestCase;

public class CalcTest extends TestCase{
    public void testAdd(){
        Calc c = new Calc();
        assertEquals(3, c.add(1, 2));
    }
}
```

TDDによる四則演算クラスの作成(6/13)

- テストケースクラスをコンパイルする。
  - Calcクラスを実装していないのでエラーになる

```
c:\angya\calc>javac -classpath .;c:\junit\junit.jar CalcTest.java
CalcTest.java:5: シンボルを解釈処理できません。
シンボル: クラス Calc
位置    : CalcTest の クラス
        Calc c = new Calc();
        ^
CalcTest.java:5: シンボルを解釈処理できません。
シンボル: クラス Calc
位置    : CalcTest の クラス
        Calc c = new Calc();
                     ^
エラー 2 個
```

TDDによる四則演算クラスの作成(7/13)

- プロダクトクラスを作成する
  - コンパイルするのに十分なコードを書く
  - 「return 0;」に着目

c:¥angya¥calc¥Calc.java

```
public class Calc{
    public int add(int a, int b){
        return 0;
    }
}
```

TDDによる四則演算クラスの作成(8/13)

- プロダクトクラスをコンパイルする

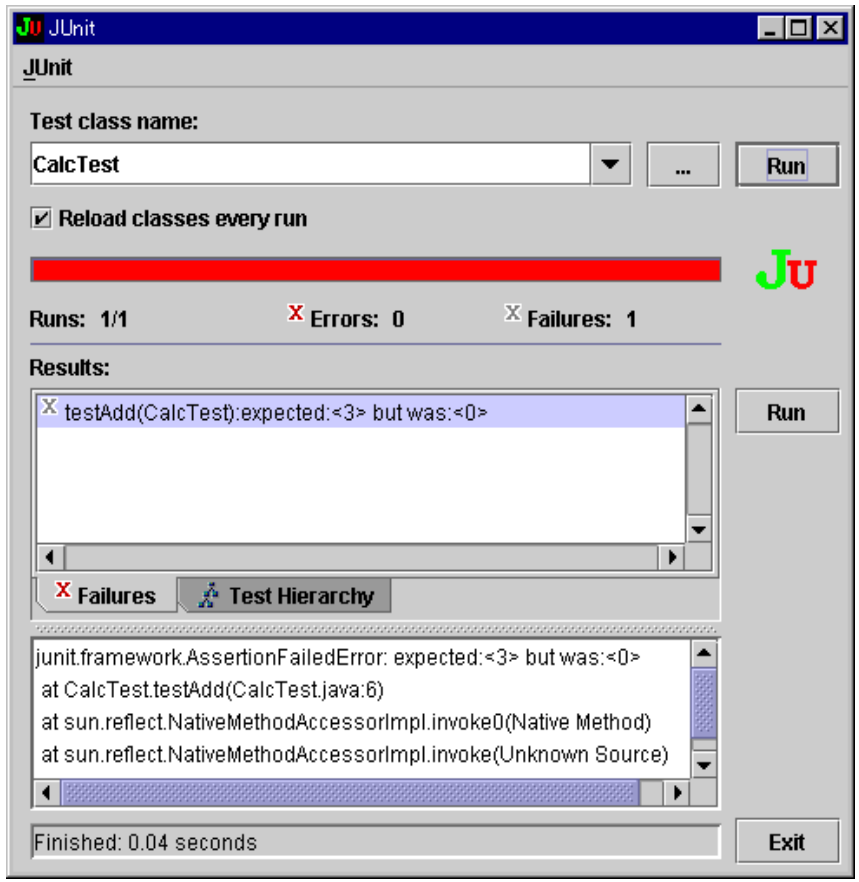
```
c:\angya\calc>javac -classpath .;c:\junit\junit.jar Calc.java
```

- テストケースクラスをコンパイルする

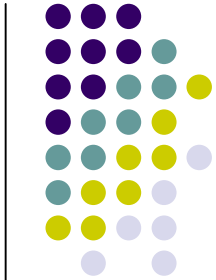
```
c:\angya\calc>javac -classpath .;c:\junit\junit.jar CalcTest.java
```

TDDによる四則演算クラスの作成(9/13)

- テストケースを実行する
  - 起動しているJUnitの[Run]ボタンをクリックする
  - 「add」メソッドの戻り値が0固定なので、テストは失敗する
  - これでテストが実行されていることは確認できた



TDDによる四則演算クラスの作成 (10/13)



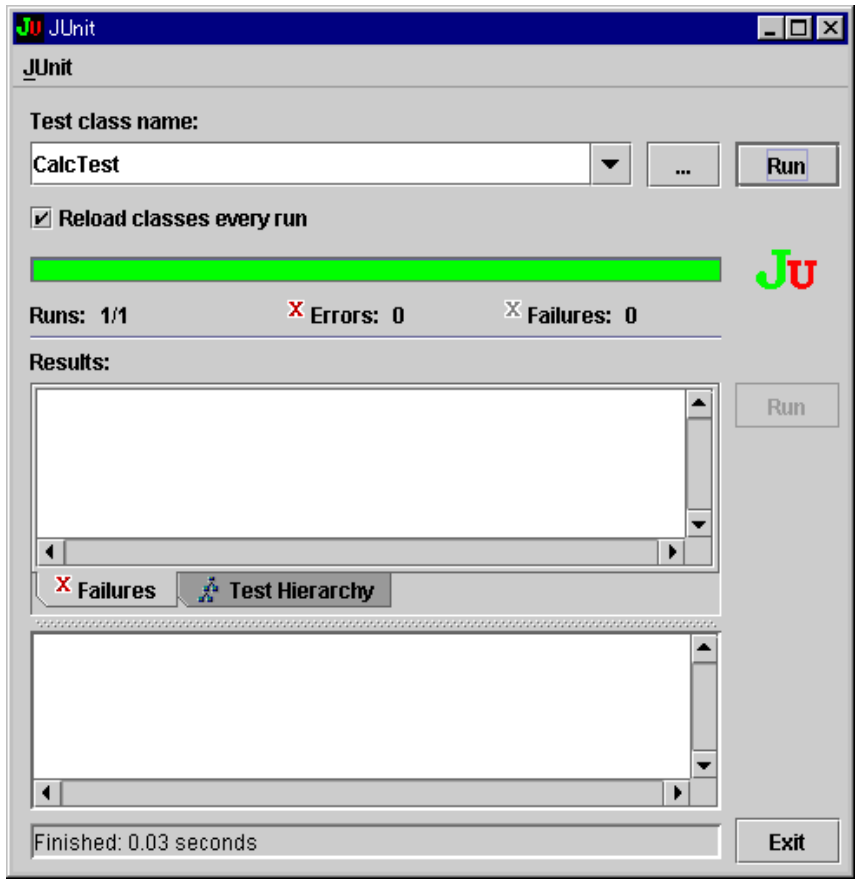
- **テストをパスする必要最小限のプロダクトコードを書く**
  - assertEquals(3, c.add(1, 2));
  - 「return 3;」**が最もシンプル**

c:¥angya¥calc¥Calc.java

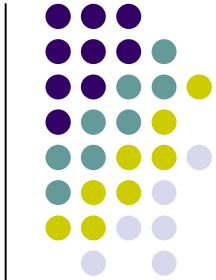
```
public class Calc{
    public int add(int a, int b){
        return 3;
    }
}
```

TDDによる四則演算クラスの作成 (11/13)

- プロダクトクラスをコンパイルし、テストケースを実行する
  - テストケースが正しいことが確認できた



TDDによる四則演算クラスの作成 (12/13)

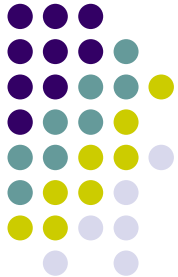


- **リファクタリング**
  - **重複を取り除く**
  - assertEquals(3, c.add(1, 2));
  - return 3;
  - 「3」**が重複している**
  - 「return a + b;」**に変更**

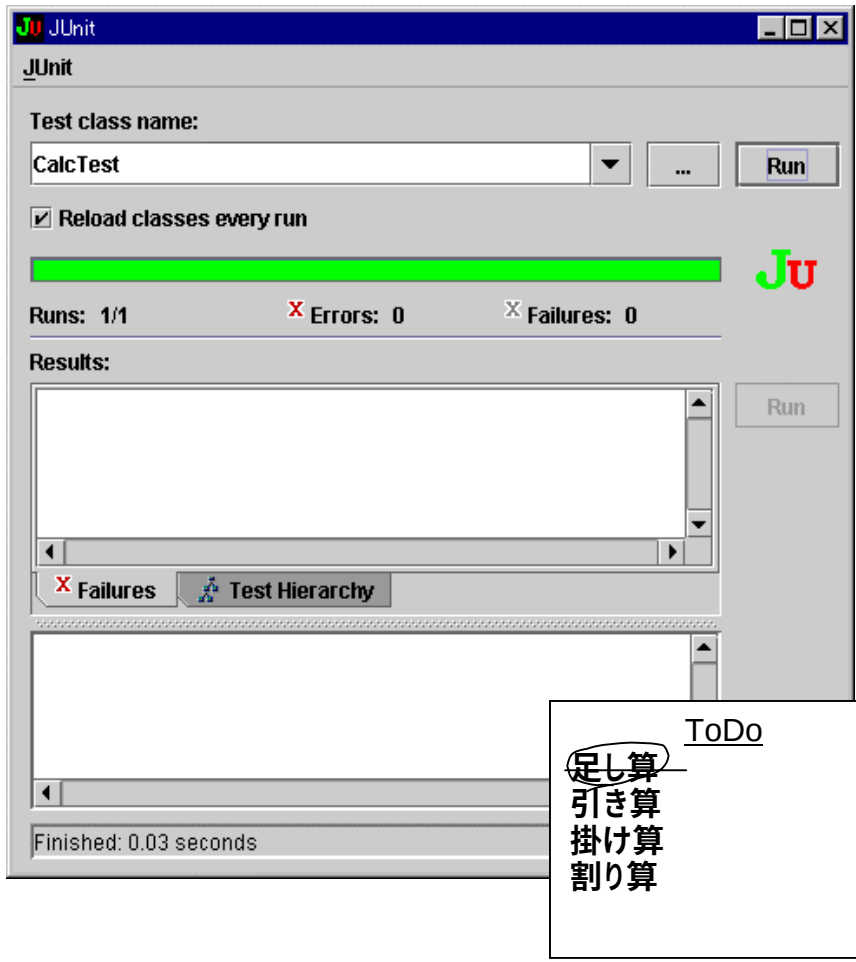
c:¥angya¥calc¥Calc.java

```
public class Calc{
    public int add(int a, int b){
        return a + b;
    }
}
```

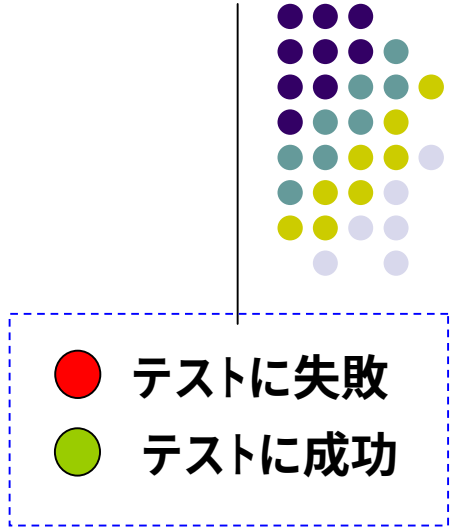
TDDによる四則演算クラスの作成 (13/13)



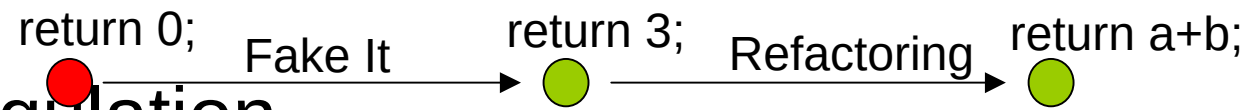
- プロダクトクラスをコンパイルし、テストケースを実行する
  - 正しくリファクタリングできたことが確認できた



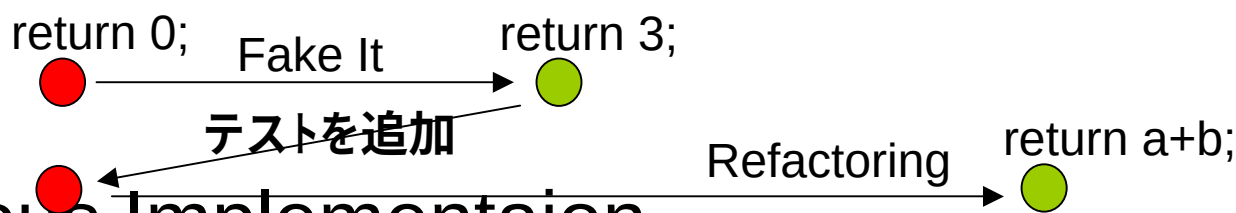
TDDの主な3つのアプローチ



- Fake It → Refactoring



- Triangulation

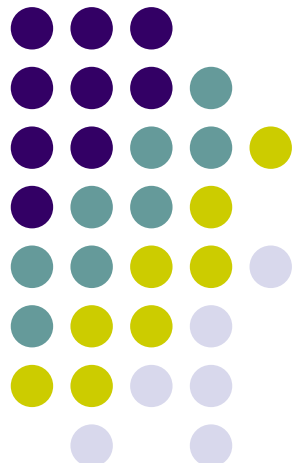


- Obvious Implementaion

$(X+Y)^2$を展開する

- 公式を知ってる人は
  - $X^2+2XY+Y^2$
- 公式を知らない人は
  - $(X+Y)(X+Y) = X(X+Y)+Y(X+Y)$
    $= X^2+XY+XY+Y^2$
    $= X^2+2XY+Y^2$

演習



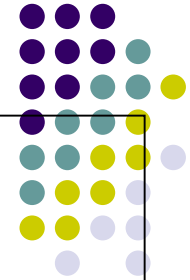
スタックの作成

- スタックの仕様
  - isEmpty()でスタックが空の場合、true。それ以外falseを返す。
  - size()でスタックのサイズを取得する。
  - push()で引数の値をスタックの一番上に積む。
    - void push(int value)
  - pop()でスタックの一番上の値を取り除く。
    - void pop()
    - スタックが空の場合、java.util.EmptyStackExceptionが発生する
  - top()でスタックの一番上の値を取得する。
    - int top()
    - スタックが空の場合、java.util.EmptyStackExceptionが発生する

スタックの作成：戦略

- **プロダクトクラス名　：**Stack
- **テストクラス名　　：**StackTest
- **進め方**
  - **テストを書いて、コンパイル。テストして。プロダクトを書いて、コンパイル。テストして。。。（つづく）**
  - **いきなりプロダクトを書いてはいけませぬ！！**
  - **まずは、**Stack**のインスタンスが作られ、**isEmpty()**が**true**を返すか確認してみては？**

解答例 (StackTestクラス)

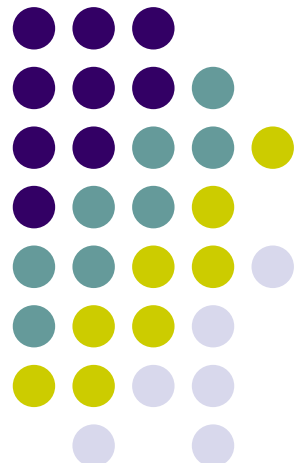


```
import java.util.EmptyStackException;
import junit.framework.TestCase;

/**
 * @author hiranabe, kitano
 */
public class StackTest extends TestCase {

    private Stack stack;
    protected void setUp() {
        stack = new Stack();
    }
    public void testCreate() {
        assertTrue(stack.isEmpty());
    }
    public void testPushAndTop() {
        stack.push(1);
        assertFalse(stack.isEmpty());
        assertEquals(1, stack.top());
        stack.push(2);
        assertEquals(2, stack.top());
    }
    public void testPushAndSize() {
        stack.push(1);
        assertEquals(1, stack.size());
        stack.push(2);
        assertEquals(2, stack.size());
    }
    public void testEmptyPop() {
        try {
            stack.pop();
            fail();
        } catch (EmptyStackException expected) {
        }
    }
    public void testPushAndPop() {
        stack.push(1);
        stack.pop();
        assertEquals(0, stack.size());
    }
    public void testPushPushPopTop() {
        stack.push(1);
        stack.push(2);
        assertEquals(2, stack.size());
        stack.pop();
        assertEquals(1, stack.top());
    }
    public void testEmptyTop() {
        try {
            stack.top();
            fail();
        } catch (EmptyStackException expected) {
        }
    }

}
```

解答例 (Stackクラス)

```
import java.util.EmptyStackException;

/**
 * @author hiranabe, kitano
 */
public class Stack {
    private int[] value = new int[10];
    private int size;
    public boolean isEmpty() {
        return size == 0;
    }
    public int top() {
        emptyCheck();
        return value[size - 1];
    }
    public void push(int value) {
        this.value[size++] = value;
    }
    public int size() {
        return size;
    }
    public void pop() {
        emptyCheck();
        --size;
    }
    private void emptyCheck() {
        if (isEmpty())
            throw new EmptyStackException();
    }
}
```

最新アジャイルテスティング



常識をくつがえすテスト記述現る?!

- 右図のような記述が動くテスト記述だったらどう思いますか？

FIT

- FIT（Framework for Integrated Test）
  - **受け入れテストのフレームワーク**
  - XP**の産みの親の一人**　Ward Cunningham**により作成され**
  - **米国**ObjectMentor**社の有志により**FITNESSE**が開発されている**

  - http://fit.c2.com/
  - http://fitnesse.org/

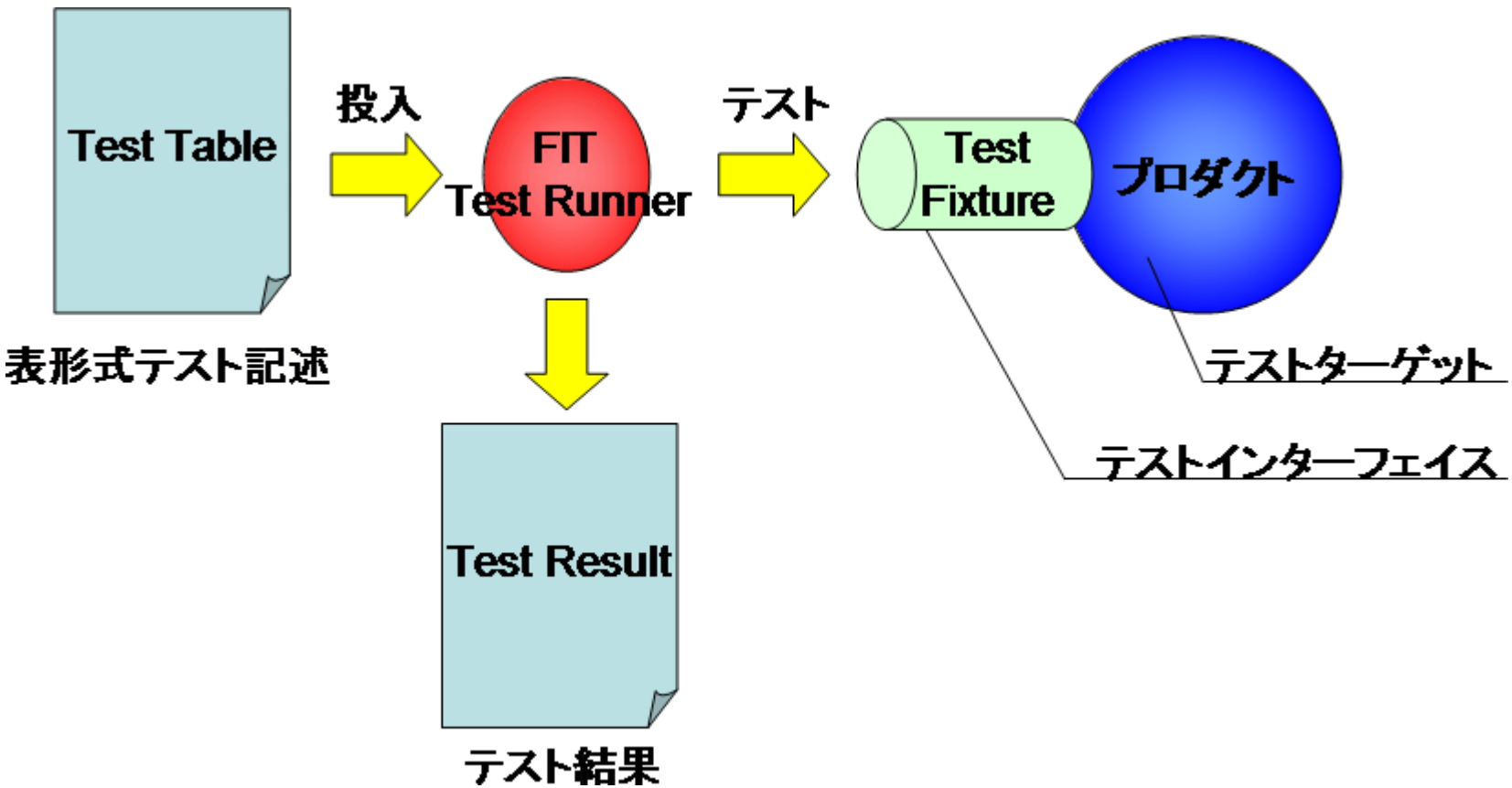
FITのメリット

- 顧客に優しいテスト
  - 顧客も書ける
    - 仕様を記述する感覚で記述できる。
  - 顧客も読める
    - 顧客と開発者との合意点が明確になる。
  - 顧客も見える
    - 受け入れテストで進捗が測りやすくまた見えやすい
- 本質的なテストに顧客が参加できる
  - View（顧客）⇔Model（開発者）
- ビジュアル的にテストしているという感覚が伝わる
  - WikiTopという新しいテスティングスタイル

FITのデメリット

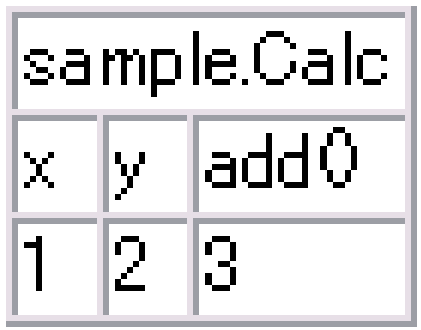
- 細かい内部仕様に関しての記述は難しい
  →JUnitで行うべき
- 例外のテストがやり難い
  →顧客に見えるインターフェイス境界のテスト（受け入れテスト）をやるべき
  →例外オブジェクトを顧客に見せる？違った形で見せれば良い

FITの構造



ポイント： FITで動かすためには、表形式で書かれている値をテストするために、TextFixtureクラスが必要となる。

FITのテスト記述



Fixtureクラス名

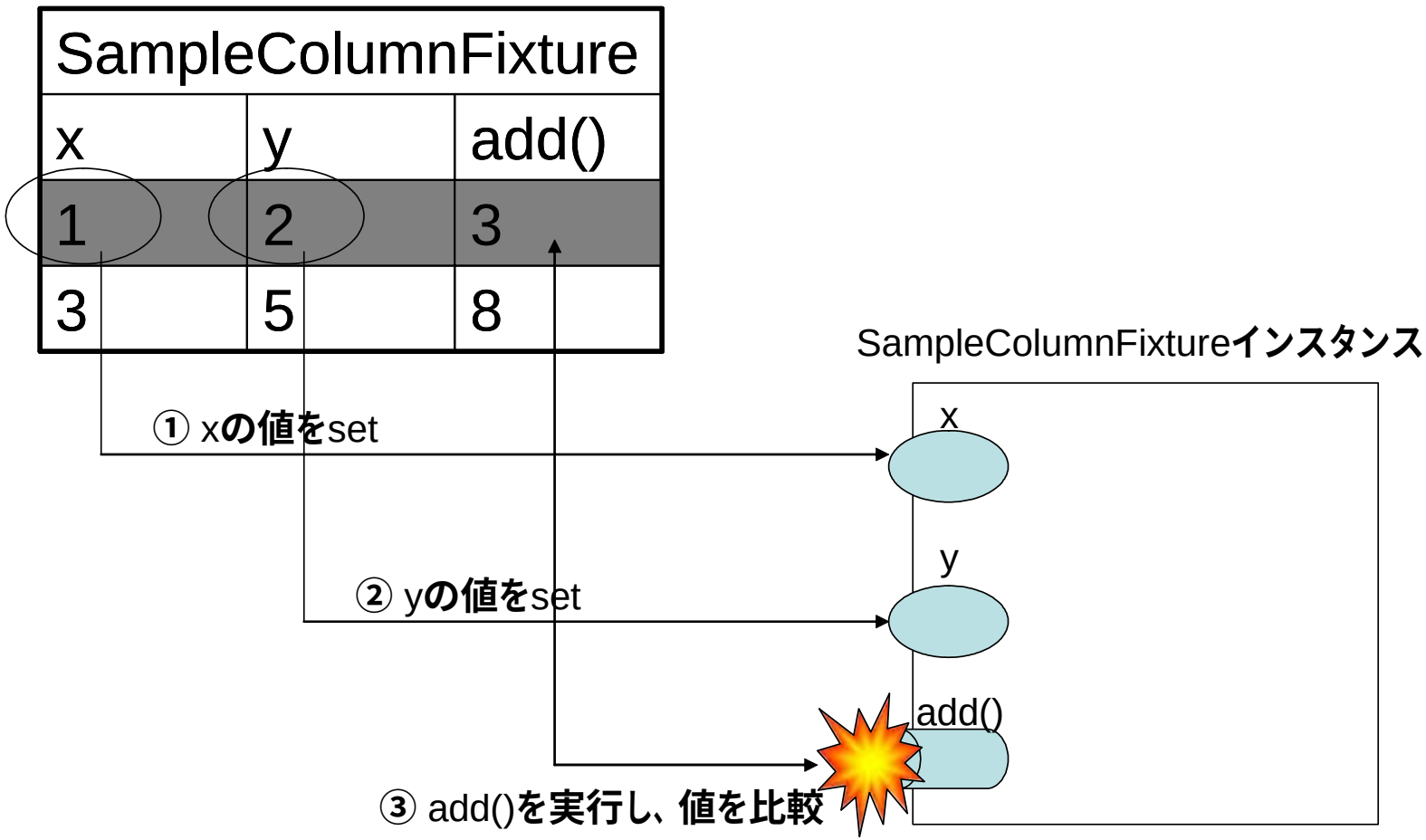
Fixtureクラスのメンバ

メンバに与える値
または比較する値

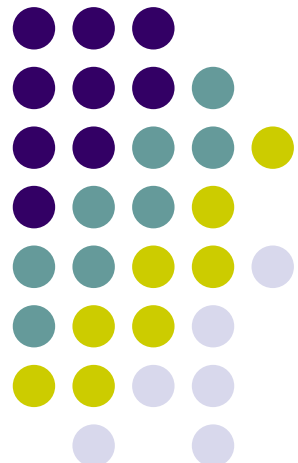
ClassPath

クラスパスを設定するページ
へのリンク

FITの動作（ColumnFixtureの場合）



FIT
デモンストレーション

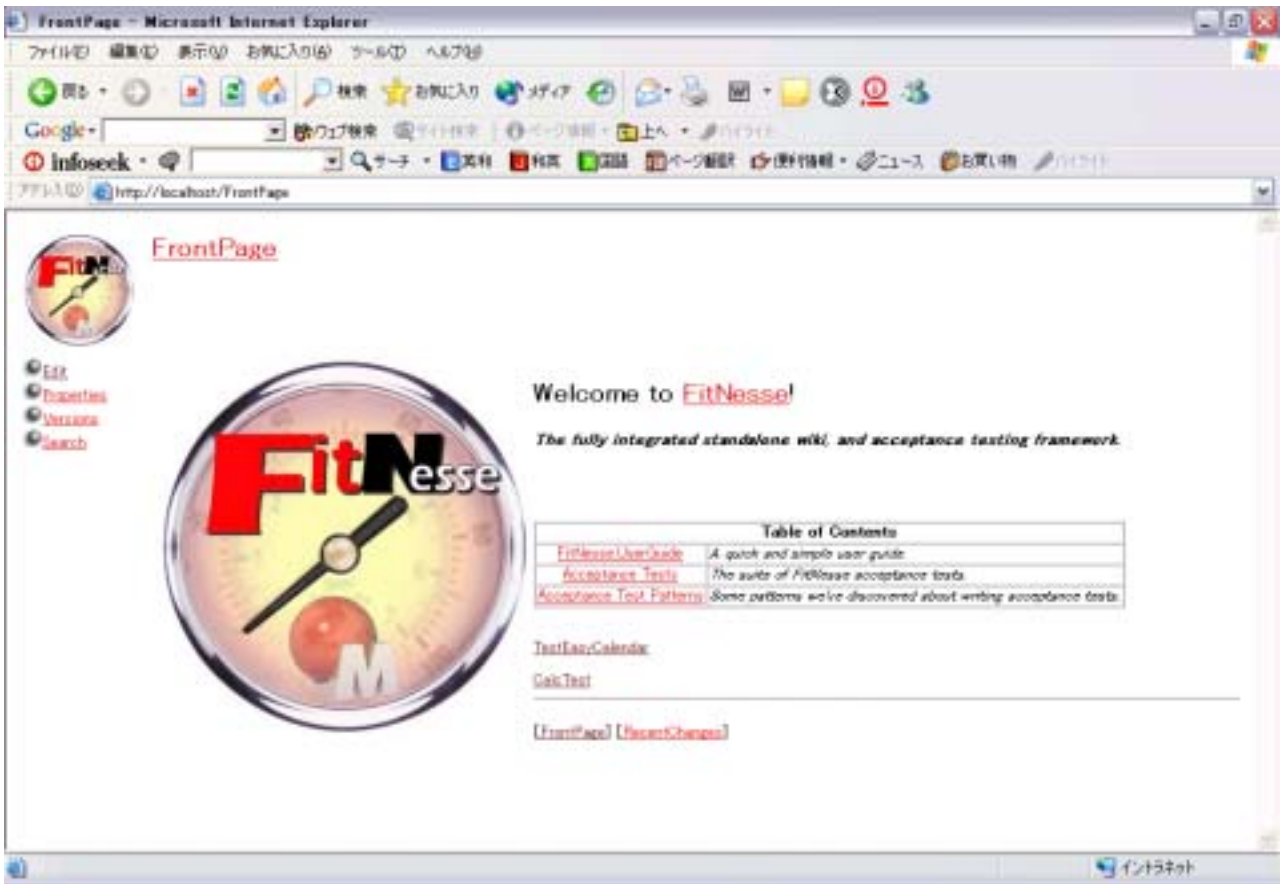


デモ0：FITインストール

- FITNESSEを自分のPCの適当なフォルダに置く
- fitnesseフォルダ/run.batを起動
  - この時Apacheなどの80ポートをListenしているプロセスは落としてください。
- Webブラウザでhttp://localhost/にアクセス

- 以上、インストール終了

デモ1：FITNESSE画面



CalcTestを押してFITを体験しましょう。

デモ2：CalcTest

CalcTest

- Test
- Edit
- Properties
- Versions
- Search
- Refactor

sample.Calc		
x	y	add()
1	2	3

ClassPath

[FrontPage] [RecentChanges]

ClassPathページでクラスパスの設定を行い、Testボタンを押してみましょう。

クラスパスの設定方法

- ClassPathページのリンクを押し、Editボタンを押し、編集します。
- 例：c:¥eclipse¥workspace¥sample

```
!path fit.jar
!path fitnesse.jar
!path C:¥eclipse¥workspace¥sample
```

- Saveボタンを押せばクラスパスの設定は完了です。

デモ3：テスト実行！！　あれ？！

sample.Calc

```
java.lang.ClassNotFoundException: sample.Calc
        at java.net.URLClassLoader$1.run(URLClassLoader.java:199)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:187)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:289)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:274)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:235)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:302)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:141)
        at fit.Fixture.doTables(Fixture.java:69)
        at fitnesse.FitFilter.process(Unknown Source)
        at fitnesse.FitFilter.run(Unknown Source)
        at fitnesse.FitFilter.main(Unknown Source)
```

x	y	add()
1	2	3

現状ではまだクラスがないので、当然です。クラスを作ってみましょう。

デモ4：Calcクラス作成

- **エディタやEclipseなどを利用して、**sample.Calc**クラスを作ります。**

```
package sample;

import fit.ColumnFixture;

public class Calc extends ColumnFixture {

}
```

これでもテストは失敗します。そのエラー（＝成功するためのガイド）に従い、実装していくのです。

デモ5：テスト成功

```
package sample;

import fit.ColumnFixture;

public class Calc extends ColumnFixture {

        public int x;
        public int y;
        public int add() {
                return x+y;
        }
}
```

sample.Calc		
x	y	add()
1	2	3

FIT演習

- FITNESSEのFrontPageに戻り、「TestEasyCalendar」を押して課題に取り組みましょう。

課題：簡易カレンダー

EasyCalendarFixture?		
from	to	days()
2001/01/10	2001/01/20	10
2001/01/10	2001/02/10	30
2001/01/10	2002/01/10	360
2001/01/10	2002/12/25	705

・日付計算の簡略化のため、1ヶ月を30日、1年を360日として扱うカレンダーとする。
・カラムとしては、from, to, days()。
・days()は、fromからtoまでの経過日数を計算する。

課題：実装例

EasyCalendarFixtureクラス（テストインターフェイスクラス）

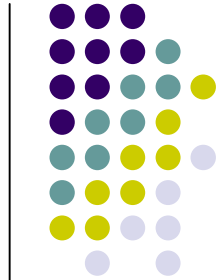
```
import java.util.Date;
import fit.ColumnFixture;

public class EasyCalendarFixture extends ColumnFixture {
        public Date from;
        public Date to;
        public int days() {
            EasyCalendar ezcal = new EasyCalendar();
            return ezcal.getBetweenDays(from, to);
        }
}
```

EasyCalendarクラス（プロダクトクラス）

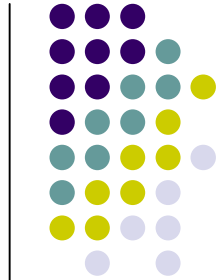
```
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class EasyCalendar {
    public int getBetweenDays(Date from, Date to) {
        Calendar calFrom = date2Calendar(from);
        Calendar calTo = date2Calendar(to);
        return getDiffYears(calFrom, calTo) * 360 +
               getDiffMonths(calFrom, calTo) *30 +
               getDiffDays(calFrom, calTo);
    }
    private int getDiffDays(Calendar calFrom, Calendar calTo) {
        return getDiffCalendar(calFrom, calTo, Calendar.DATE);
    }
    private int getDiffMonths(Calendar calFrom, Calendar calTo){
        return getDiffCalendar(calFrom, calTo, Calendar.MONTH);
    }
    private int getDiffYears(Calendar calFrom, Calendar calTo){
        return getDiffCalendar(calFrom, calTo, Calendar.YEAR);
    }
    private int getDiffCalendar(Calendar calFrom, Calendar calTo, int ymd) {
        return calTo.get(ymd) - calFrom.get(ymd);
    }
    private Calendar date2Calendar(Date date){
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(date);
        return calendar;
    }
}
```

今回のFITに関する情報は、JavaPressVol34に載ります。

Memo



Memo



Merry X’mas