DIとAOPの基礎

日本Springユーザ会
土岐孝平

JSUG

アジェンダ

- モジュールについて
- DIの実践
- DIコンテナ
- AOP
- 質疑応答

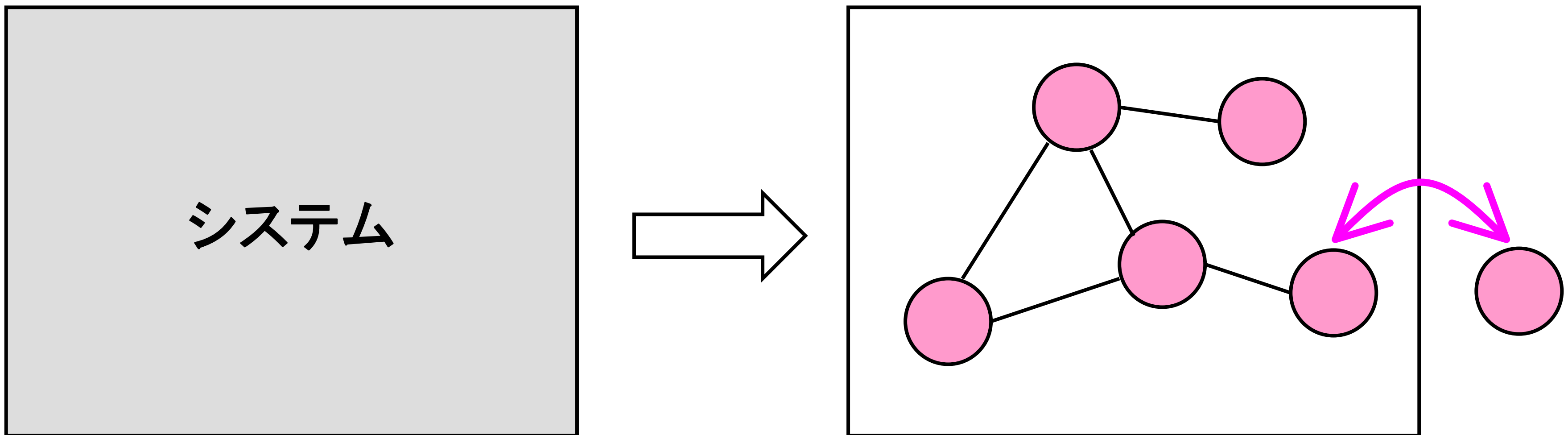
JSUG

モジュールについて

JSUG

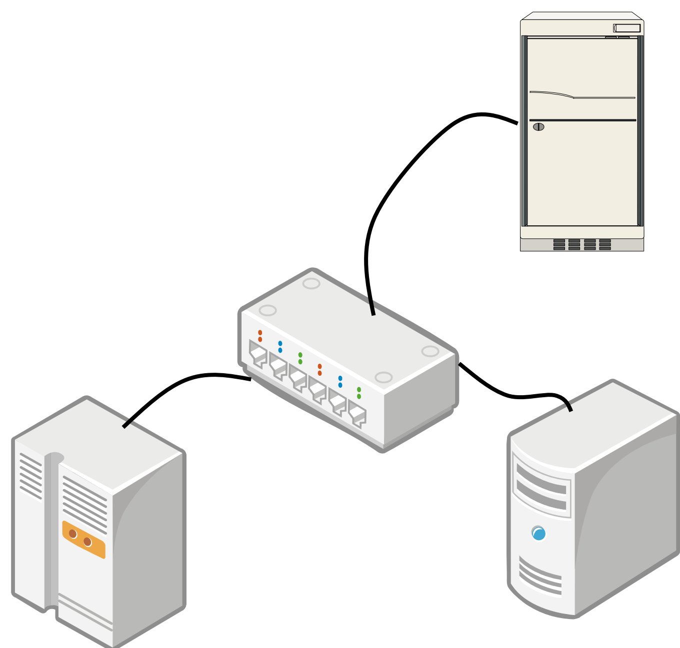
モジュールとは？

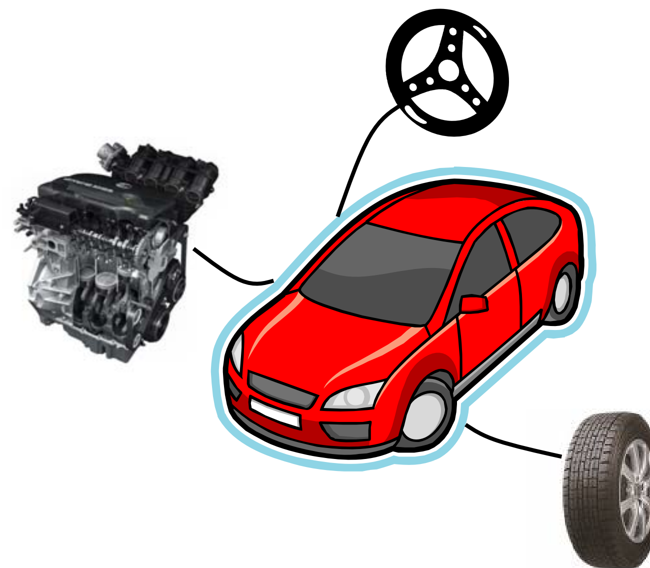
- システム(系)を構成する部品
  - 部分的な機能を持つ
- 境界部分が規格化されていて交換可能
- モジュールを組上げてシステムをつくる

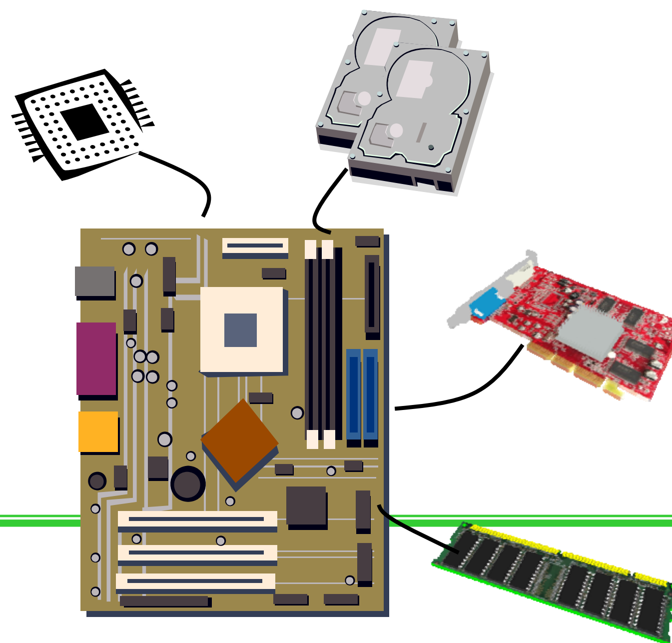


広義のモジュール

- 機能ごとに分割された部品



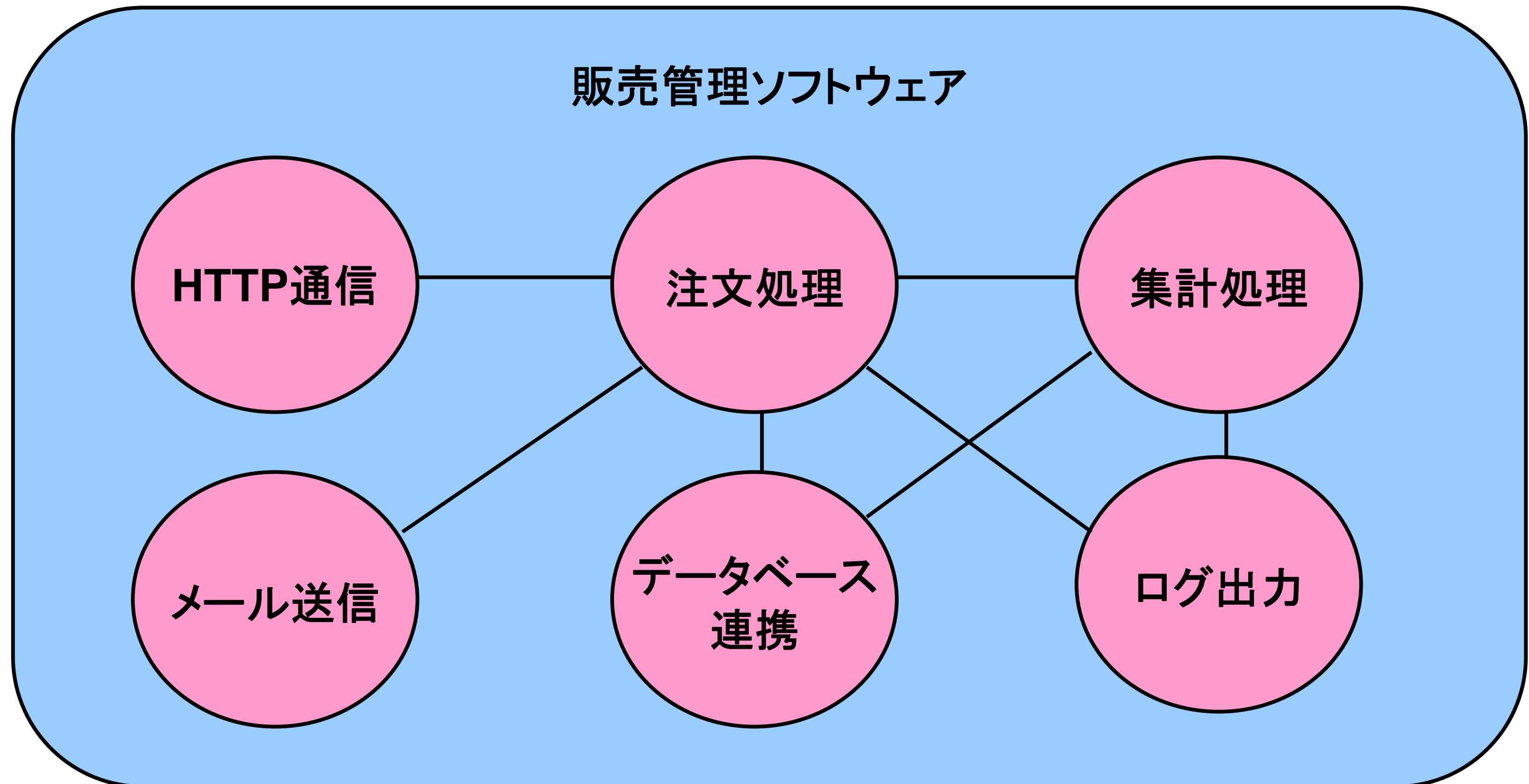




JSUG

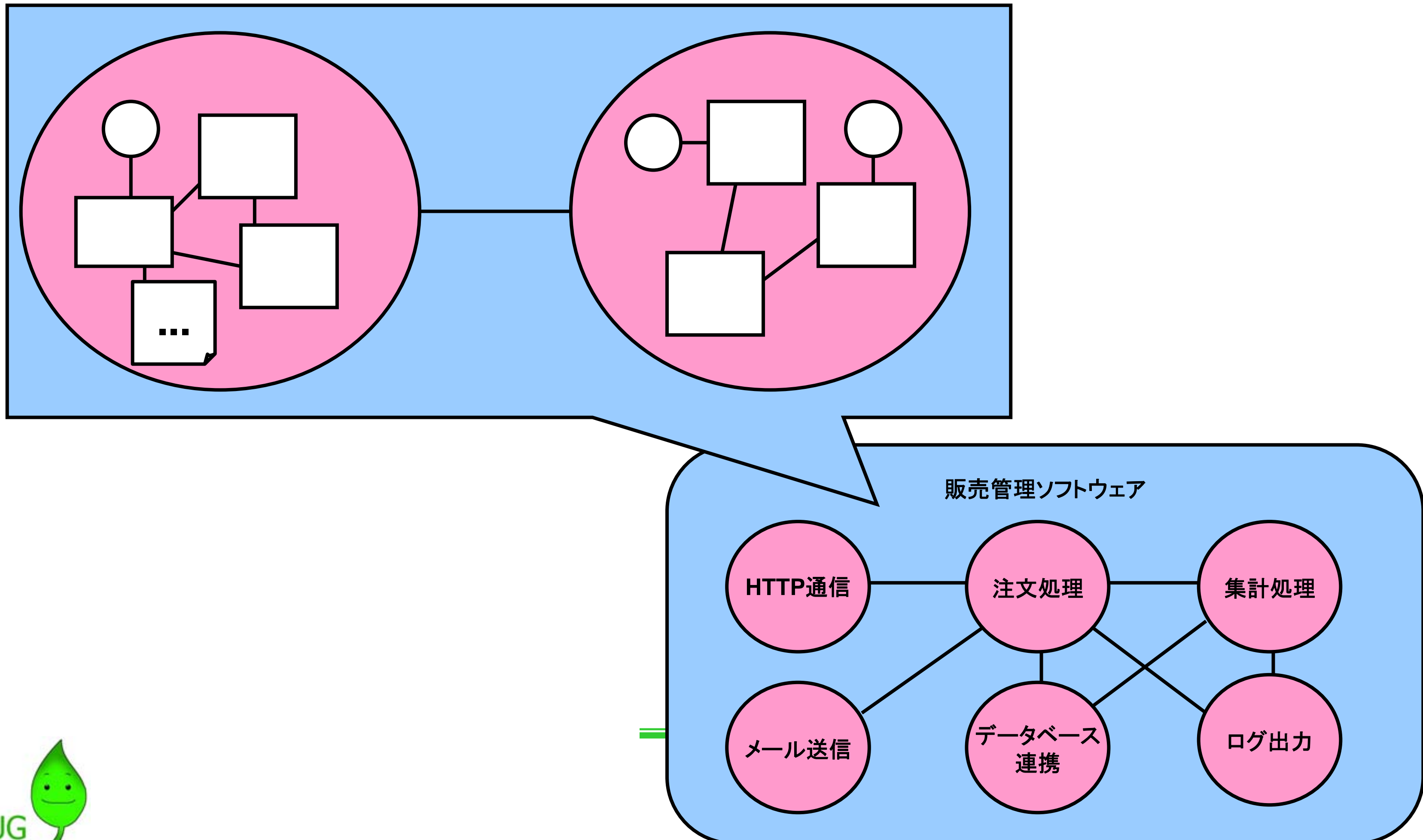
ソフトウェアのモジュール

- 機能ごとに分割されたプログラム



Javaのモジュール

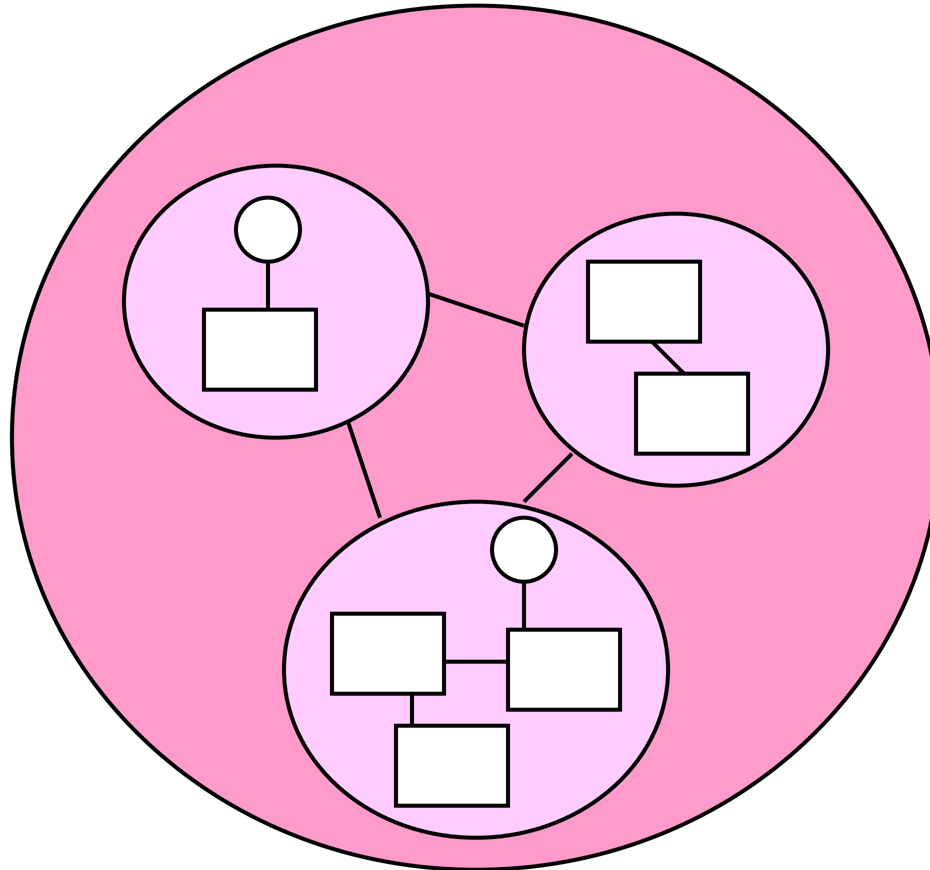
- クラス・インターフェース・外部ファイルの集合
  - ライブラリとしてjarファイルにまとめる場合もある



JSUG

Javaのモジュール

- モジュールの中にさらに小さなモジュールも･･･



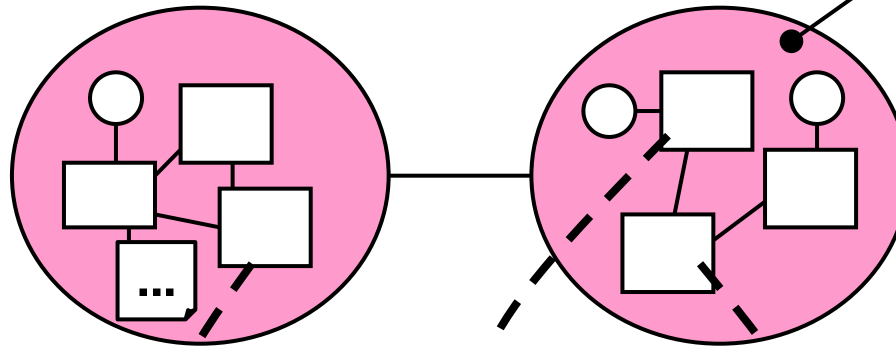
JSUG

モジュールとオブジェクト

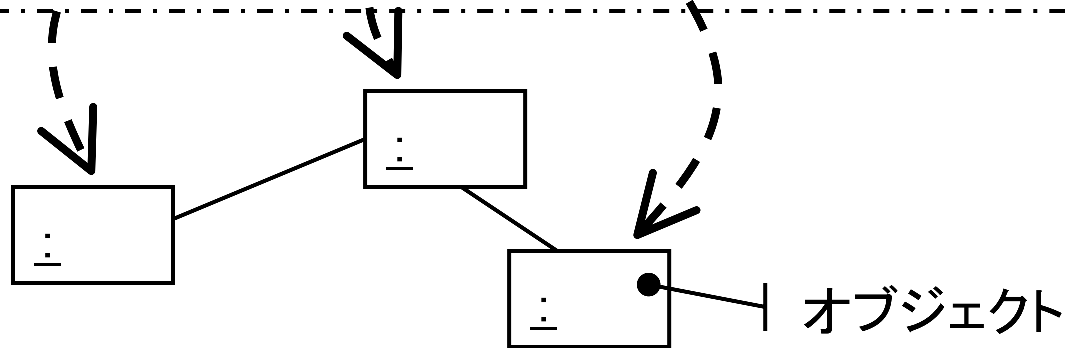
- モジュールは静的な概念
  - オブジェクトは要素に含まない

静的

クラス、外部ファイル
インターフェース



動的(実行時)



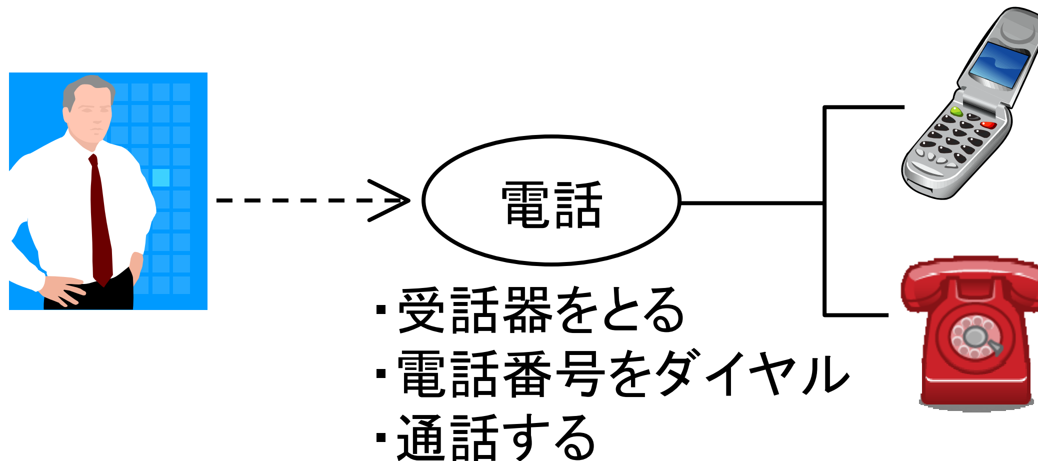
JSUG

モジュールに求められるもの

- 品質の高さ
  - 変更容易性
    - 特定のモジュールを改修・交換しても、他のモジュールに影響しない(カプセル化・ポリモーフィズム)
  - テスト容易性
    - モジュール単体でテスト可能
  - 再利用性
    - 複数のソフトウェアで使いまわせる

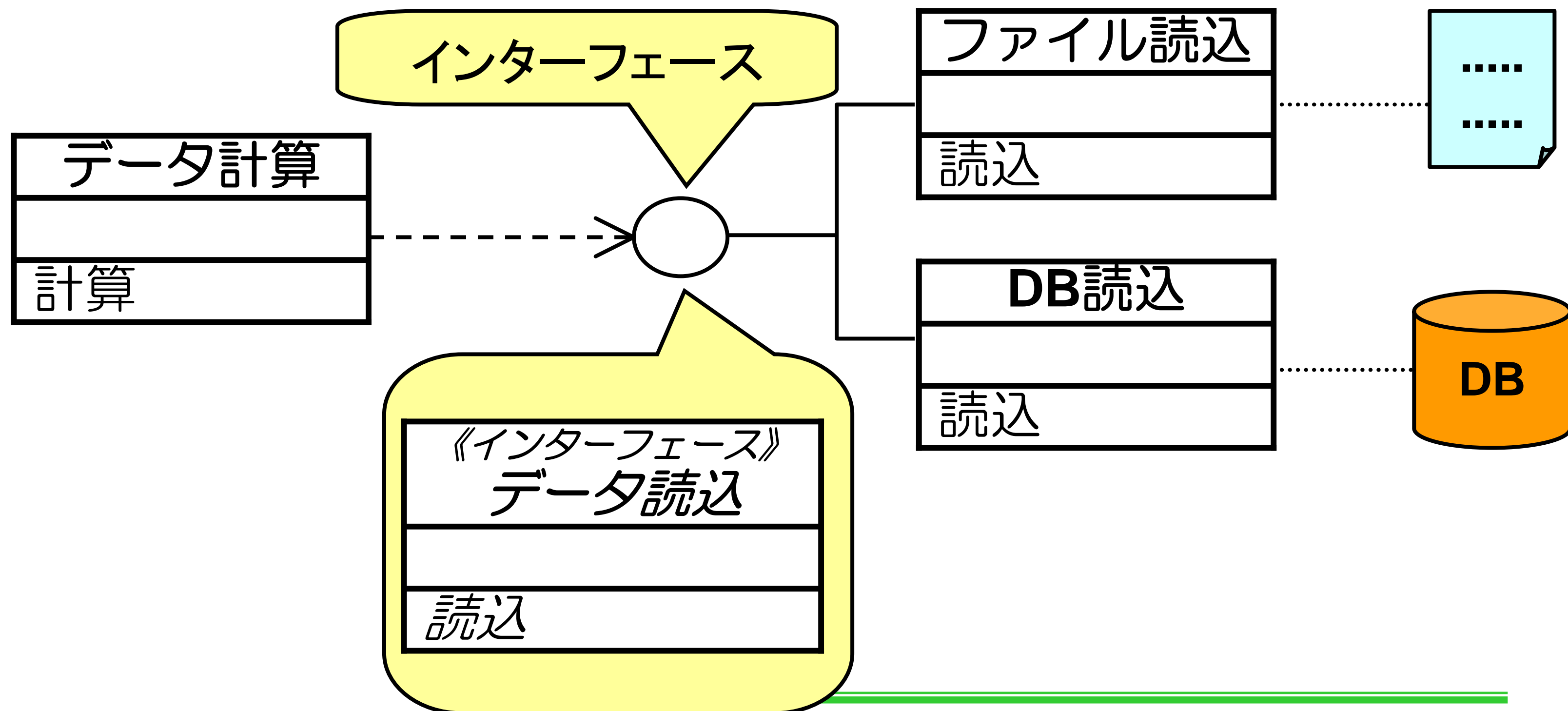
インターフェース

- ものごとの境界となる部分
  - 境界部分は規格化されている



Javaのインターフェース

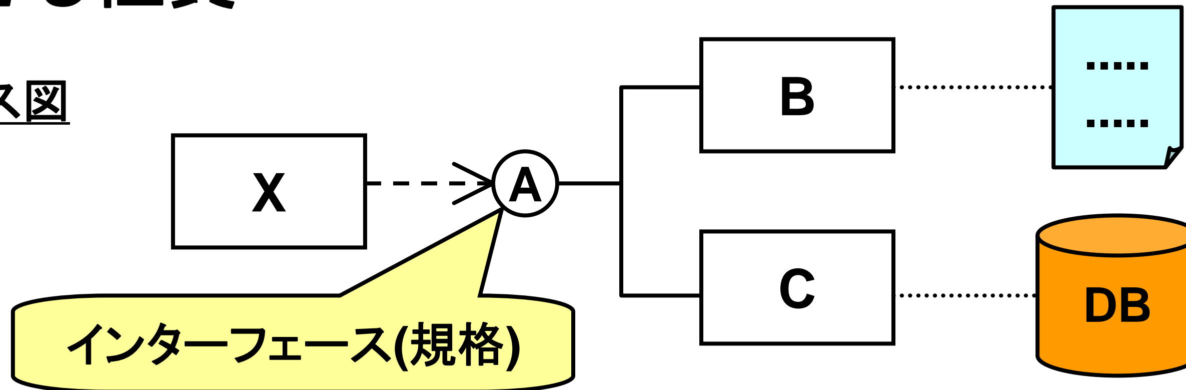
- メソッドの規格(メソッド名・引数・戻り値)だけ定義された型。実装は持たない

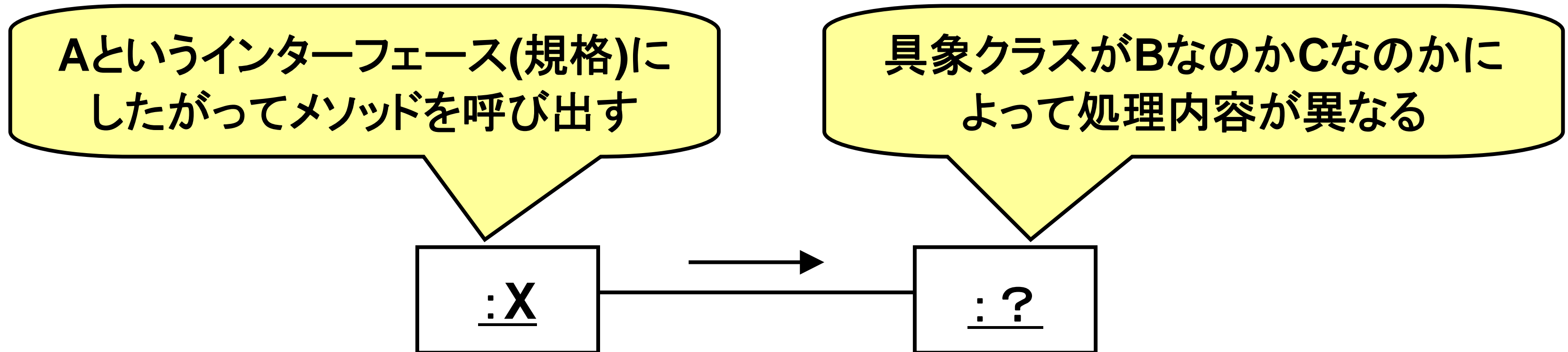


JSUG

ポリモーフィズム(多態性)

- 複数の型のオブジェクトを同一の型として扱うことができる性質

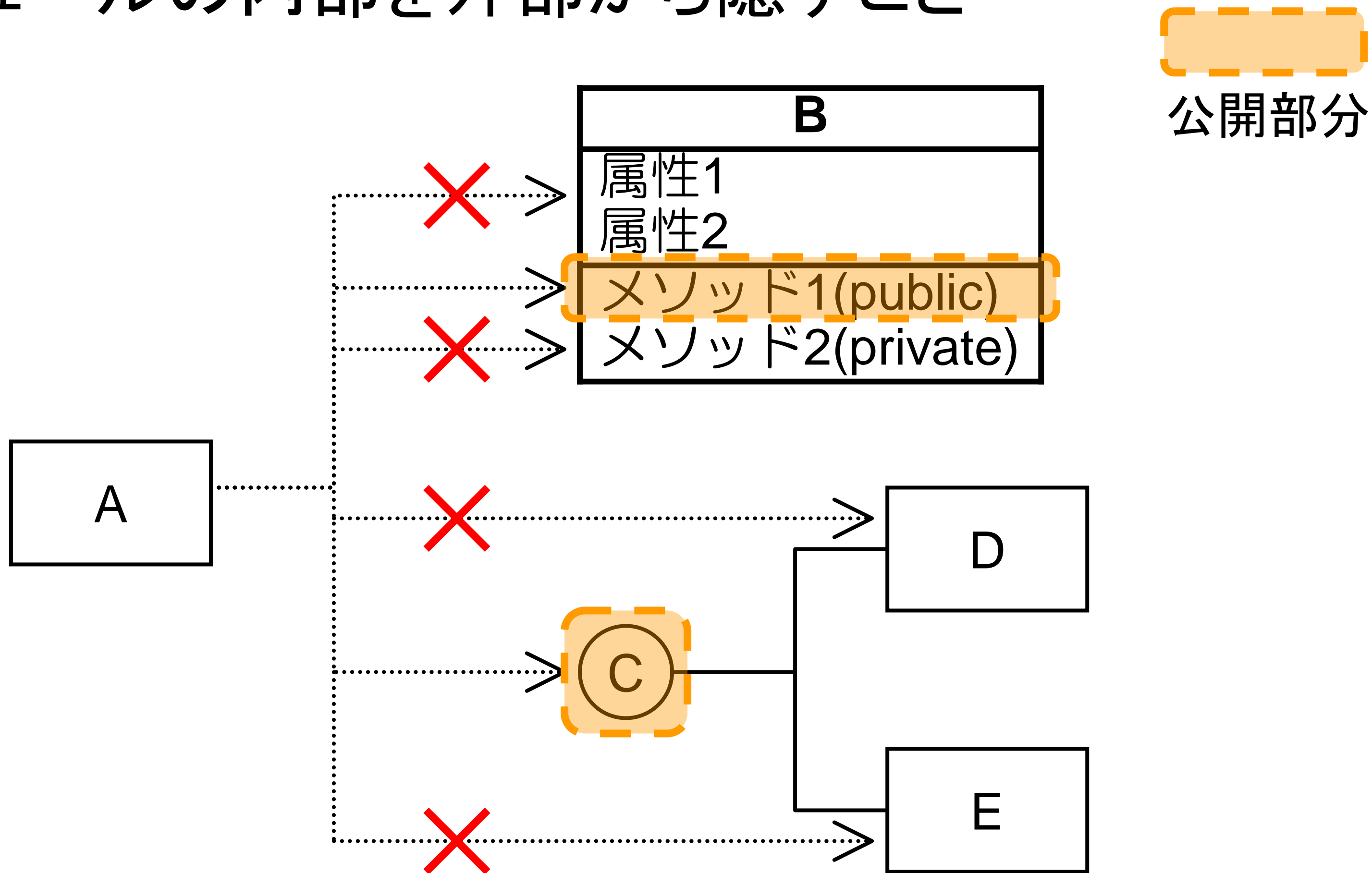




JSUG

カプセル化(隠蔽化)

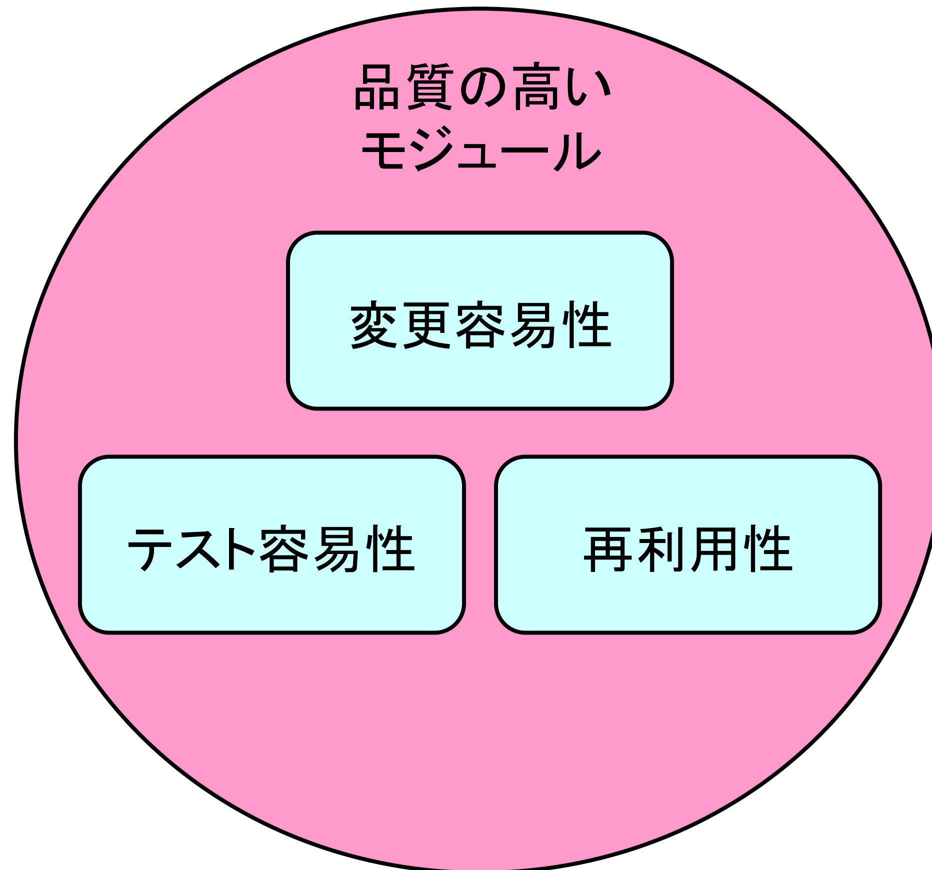
- モジュールの内部を外部から隠すこと



JSUG

モジュールとDI

- 品質の高いモジュールを作成するための設計概念



※ DI (Dependency Injection：依存性の注入)

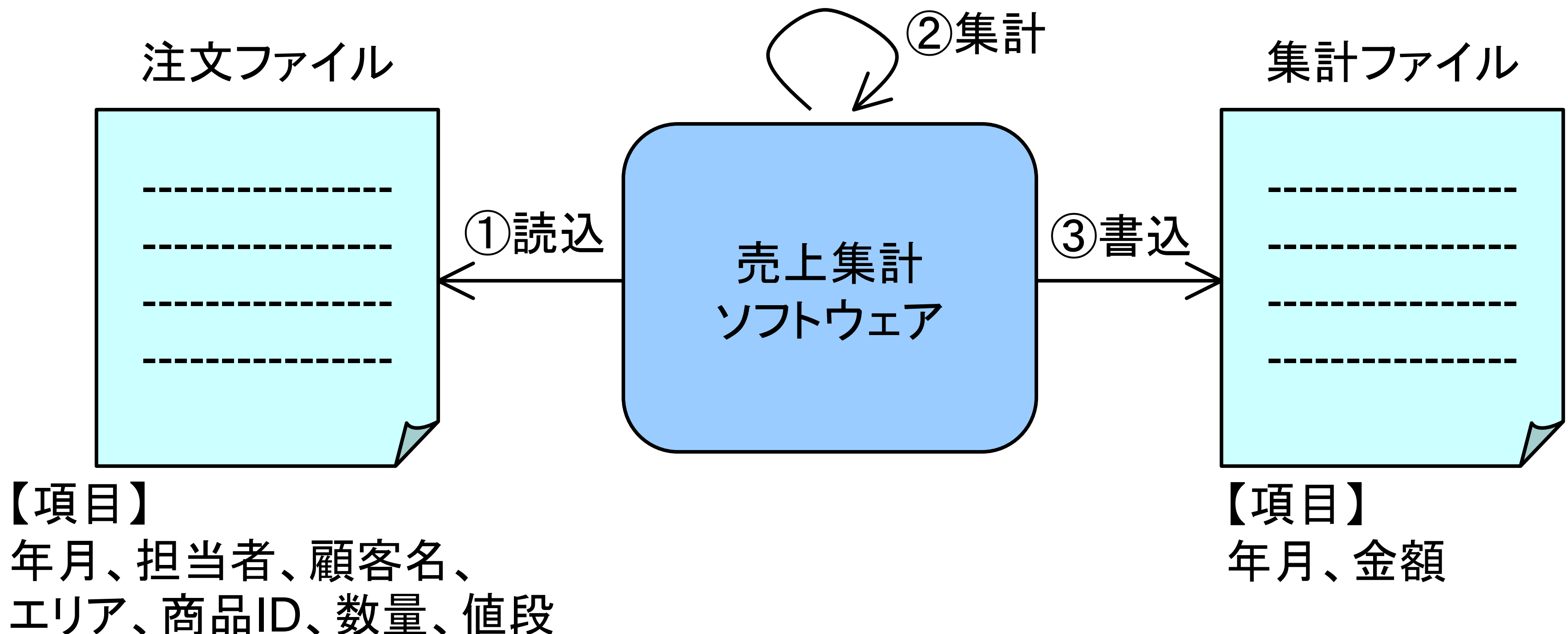
JSUG

DIの実践

JSUG

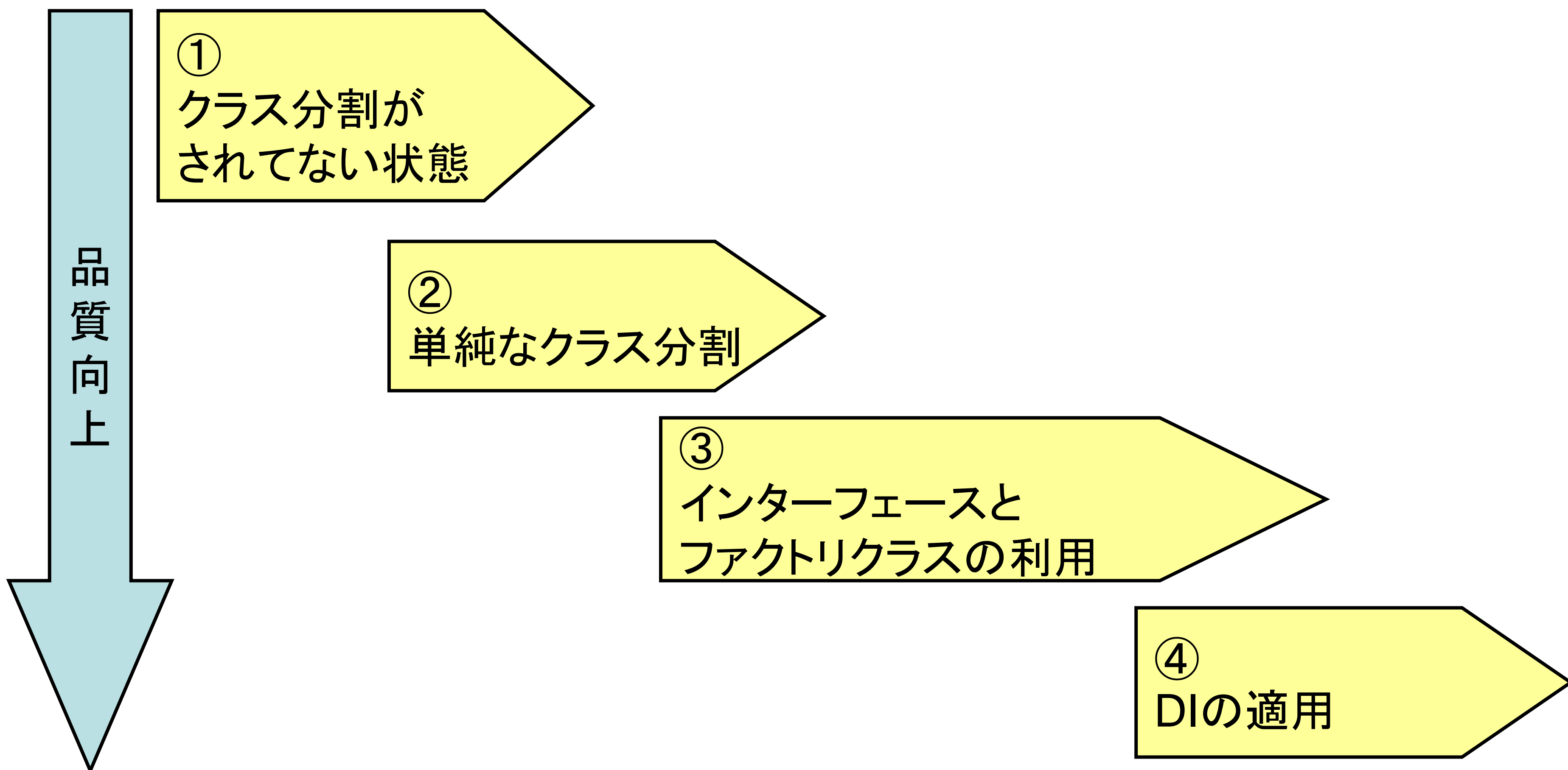
サンプルソフトウェア

- 商品の売上を集計するソフトウェア
  - 月毎の売上額を集計

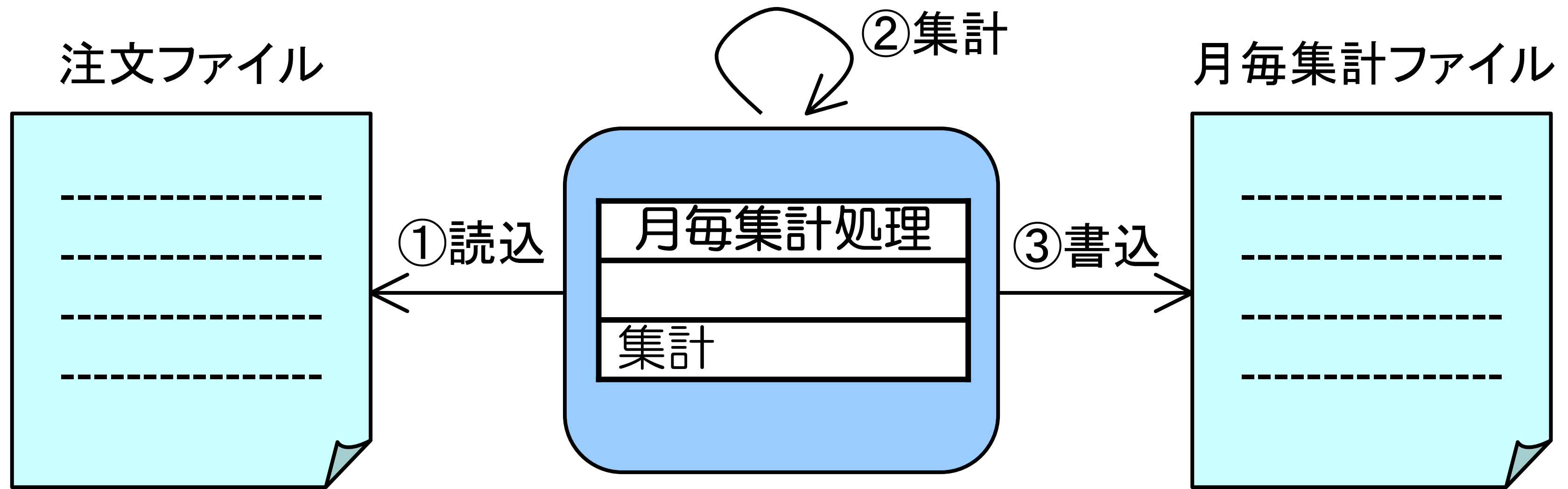


実践の流れ

- 段階を踏んでDIを適用する



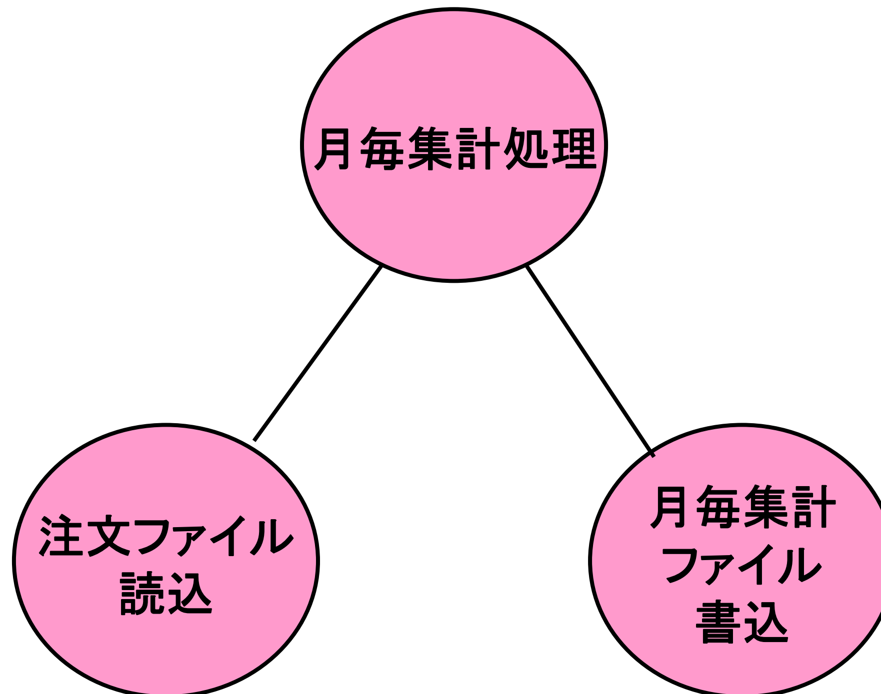
①クラス分割がされてない状態



- 担当者毎の集計が必要になったら・・・
- ファイルじゃなくてDBに読書きするケースがでてきたら・・・

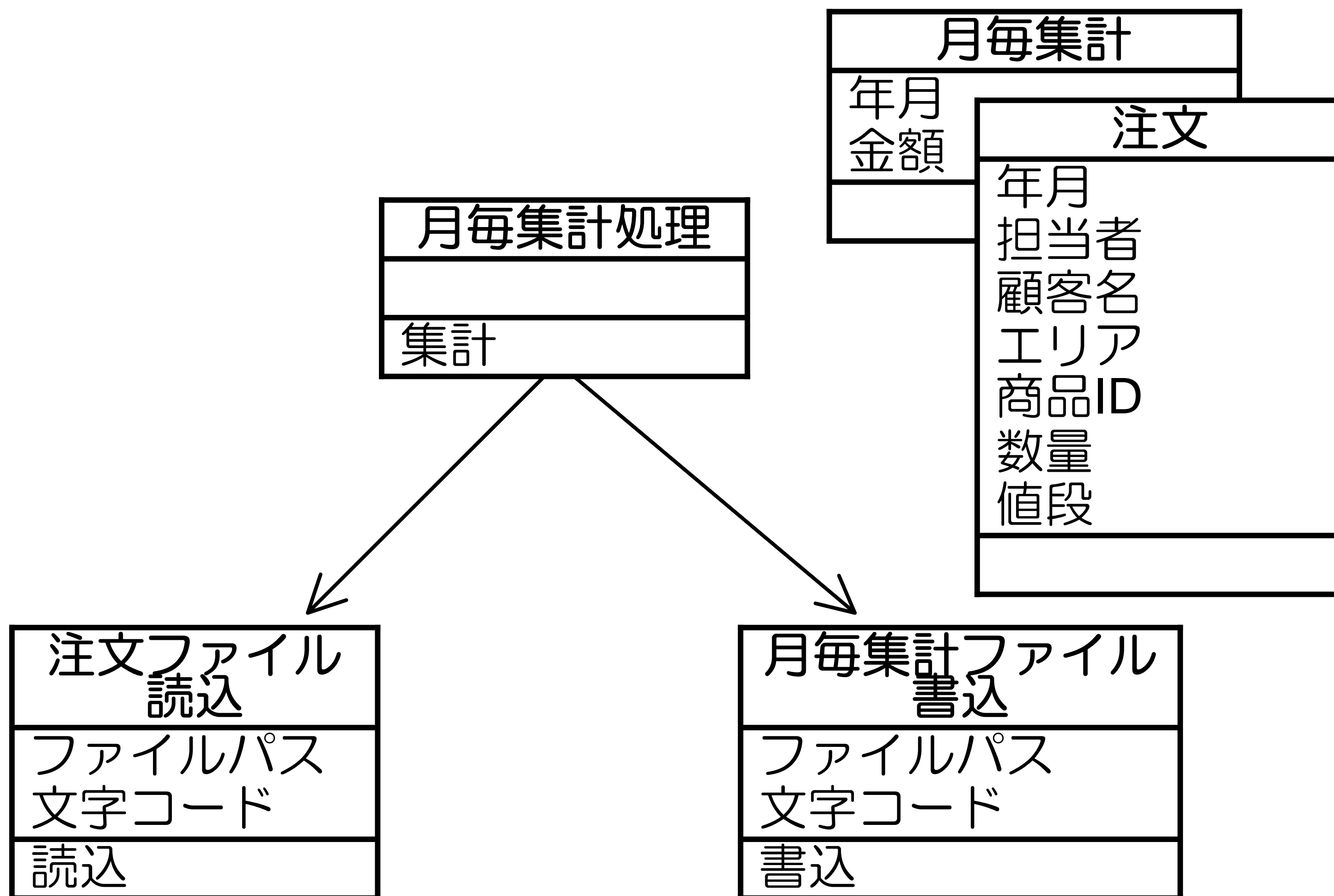
どのようにモジュールを分割するか

- テスト・再利用・変更がし易い単位



JSUG

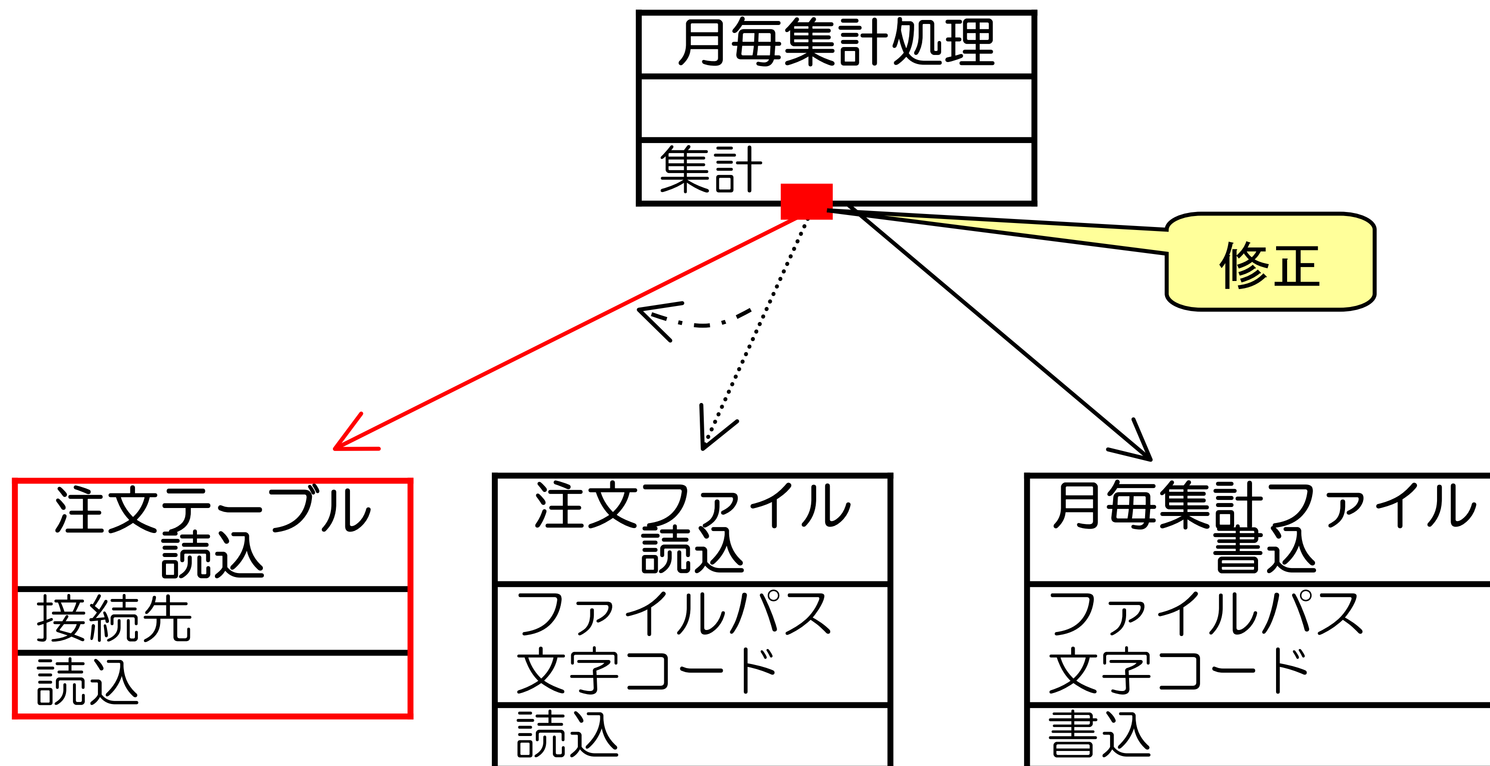
②単純なクラス分割



JSUG

変更容易性はどうか？

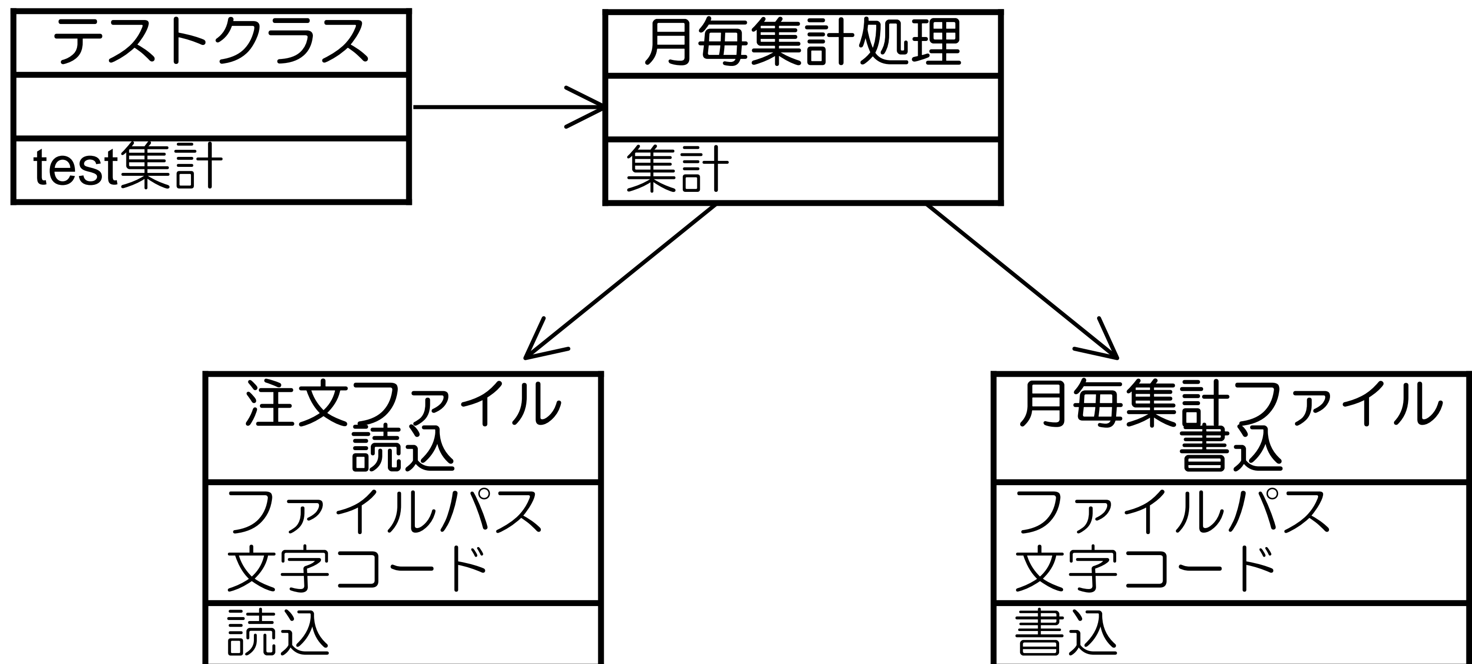
- よくない
  - 関連先を別のクラスに変更するには、関連元を修正する必要がある



JSUG

テスト容易性はどうか？

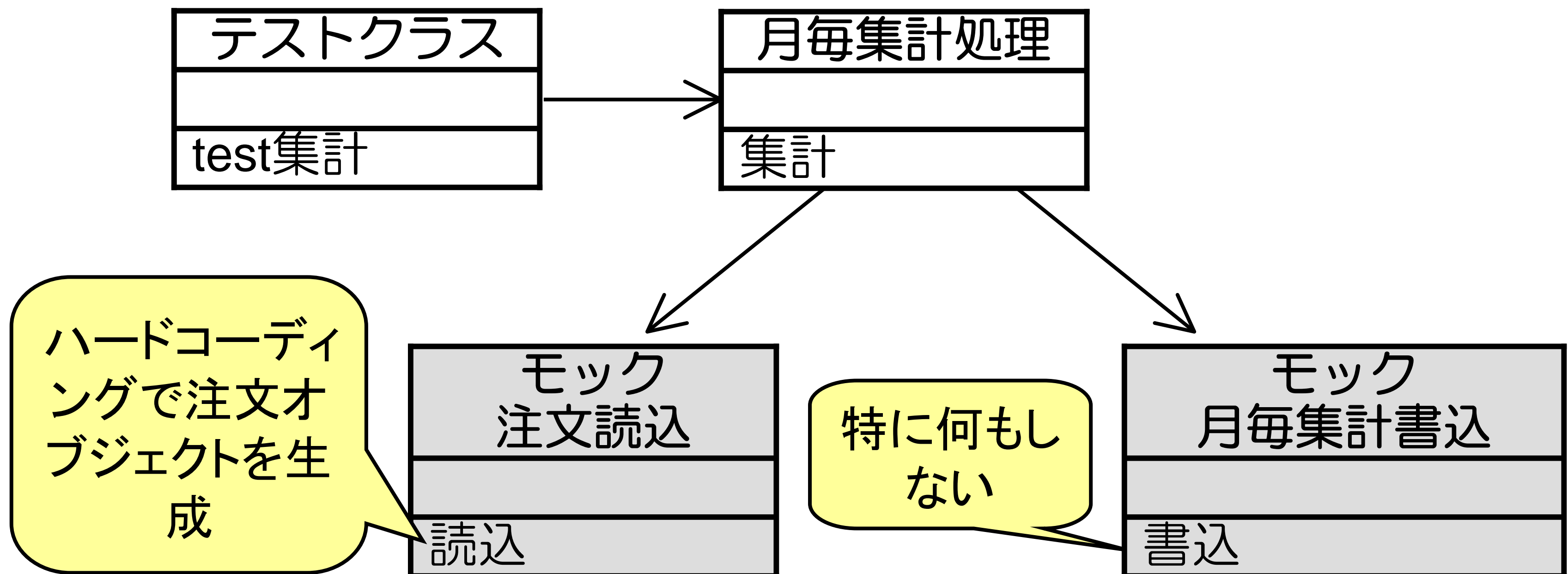
- よくない
  - 月毎集計処理クラスをテストするには、注文ファイル読込クラス・月毎集計ファイル書込クラスが必要(単体でテストできない)
    - モッククラスが使えない



JSUG

モッククラスとは？

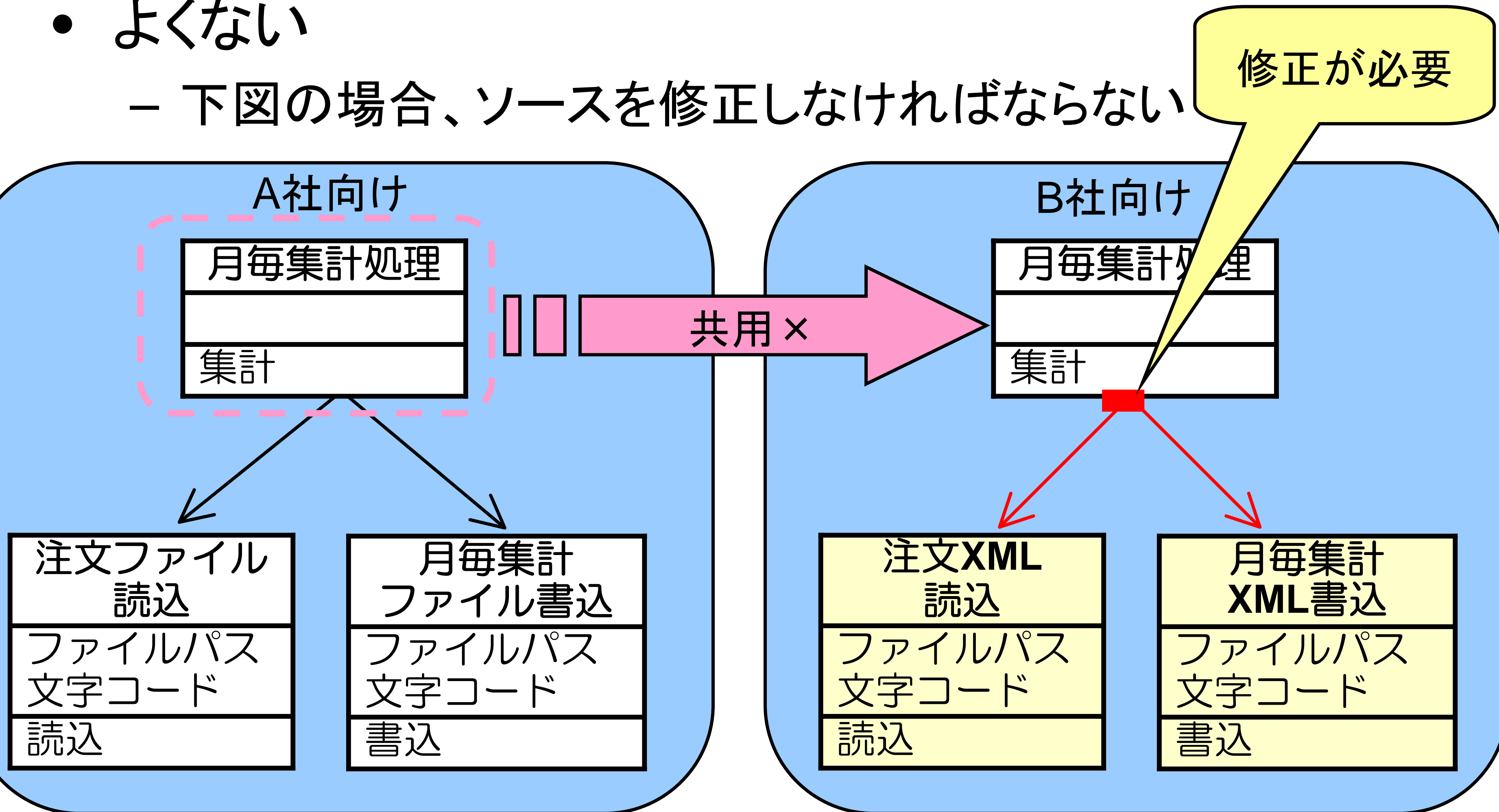
- あたかも正常に動作する偽のクラス
  - テストクラスと一緒に作成する



JSUG

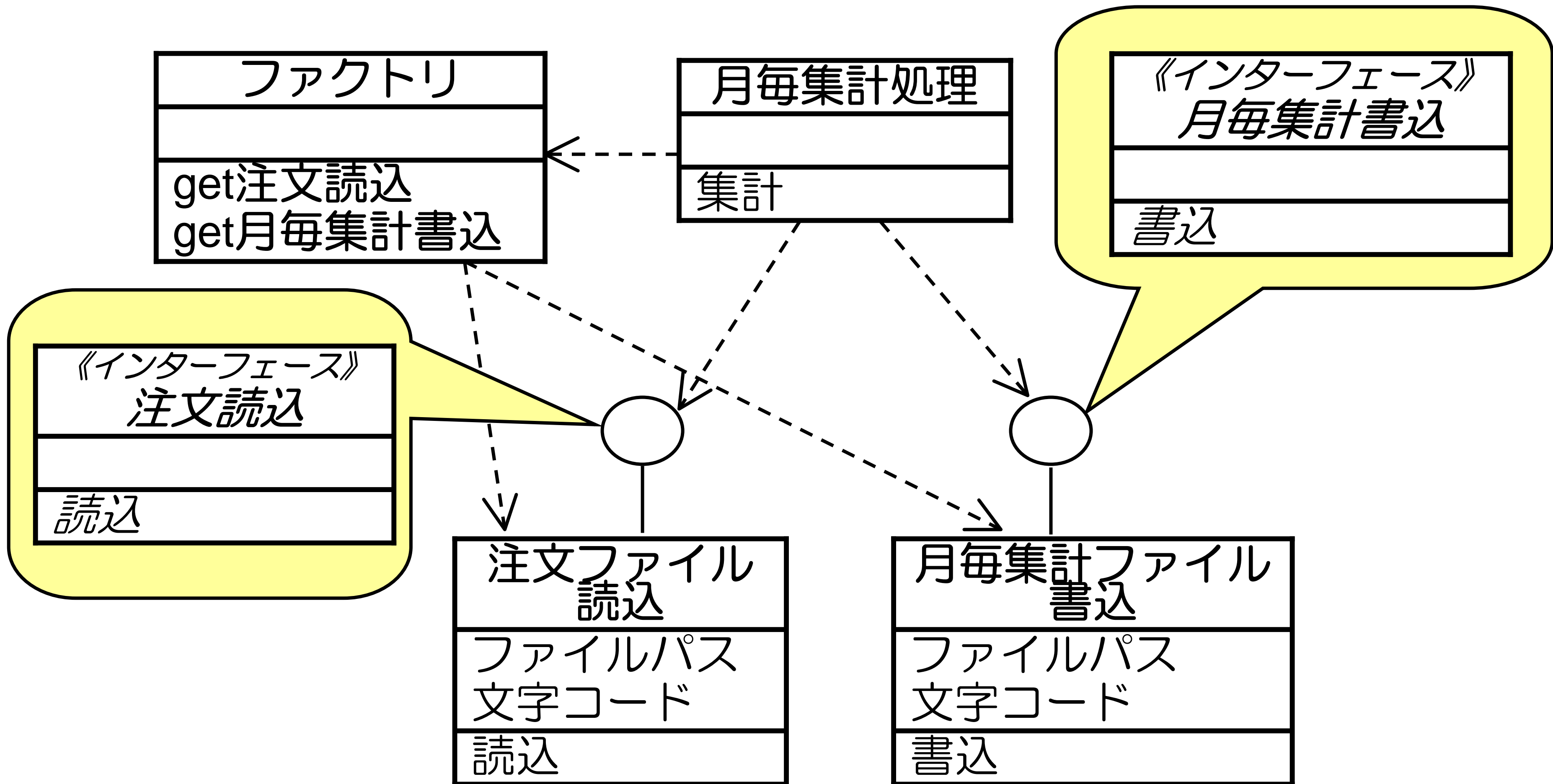
再利用性はどうか？

- よくない
  - 下図の場合、ソースを修正しなければならない



JSUG

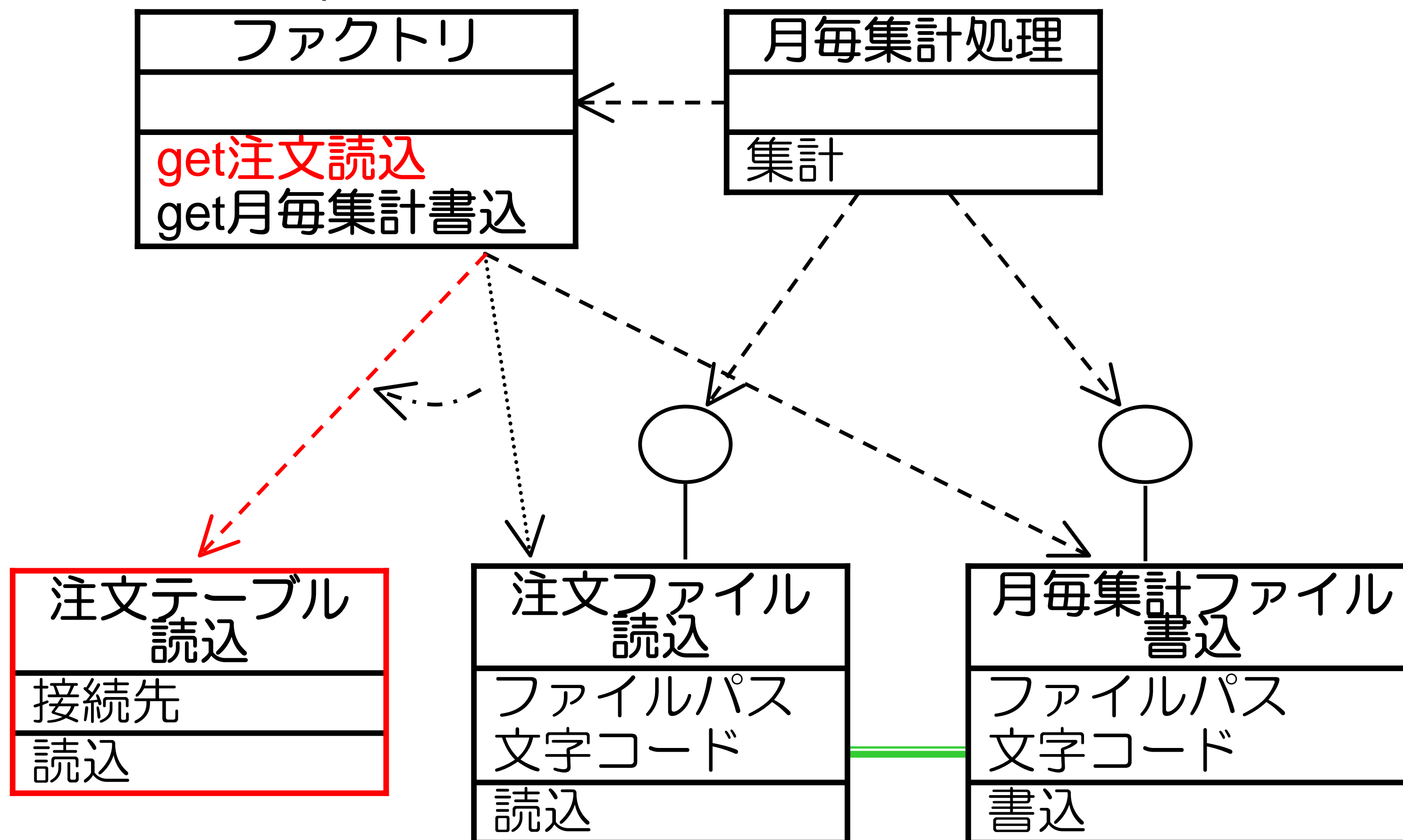
③インターフェースとファクトリクラス※の利用



※オブジェクトの生成を責務とするクラス
GOFのデザインパターンから"ファクトリ"という言葉を流用しているが、パターンを正確に適用しているわけではない

変更容易性はどうか？

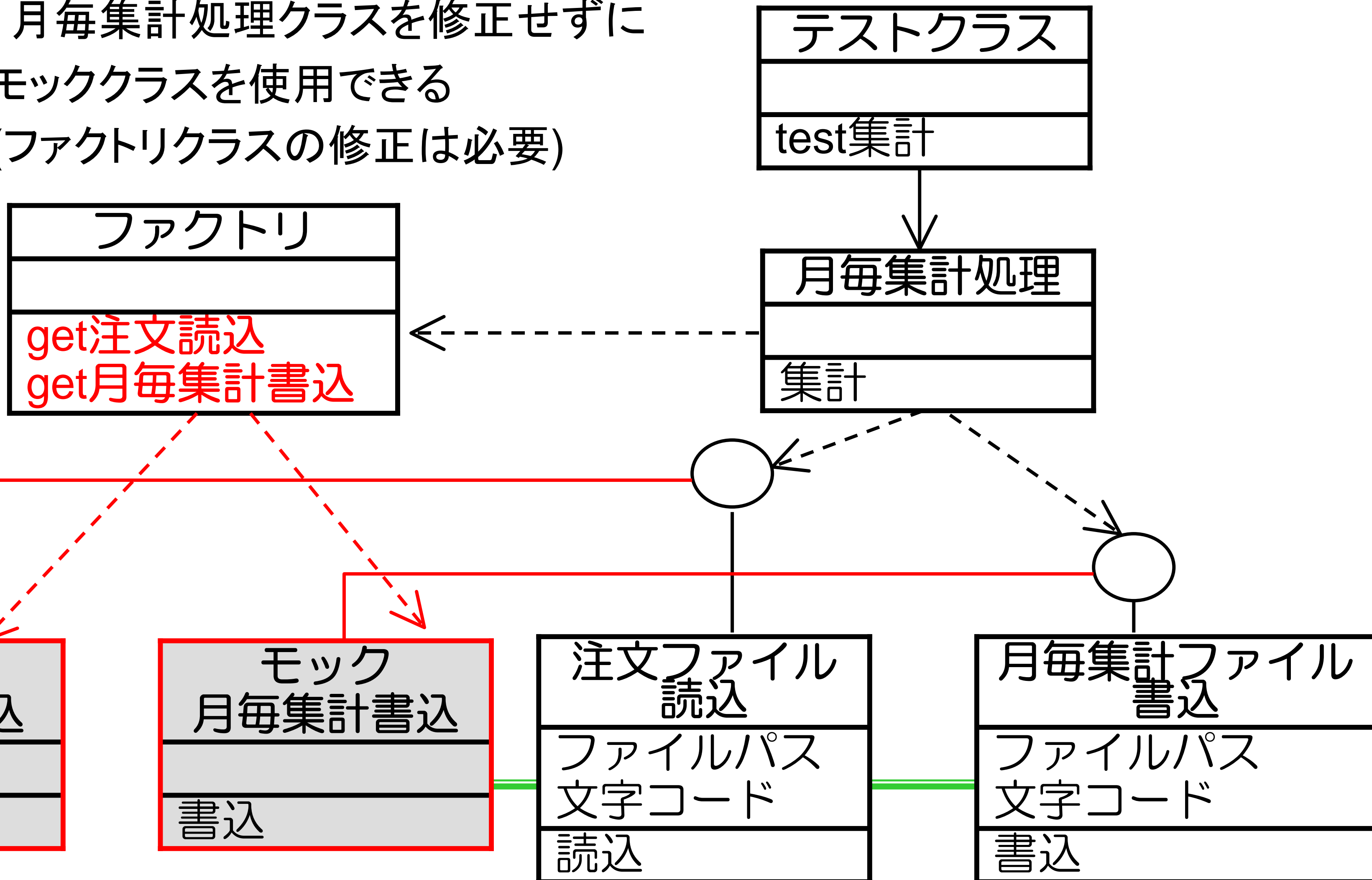
- よい
  - ファクトリクラスを修正するだけでよい(他のクラスの修正は必要ない)



JSUG

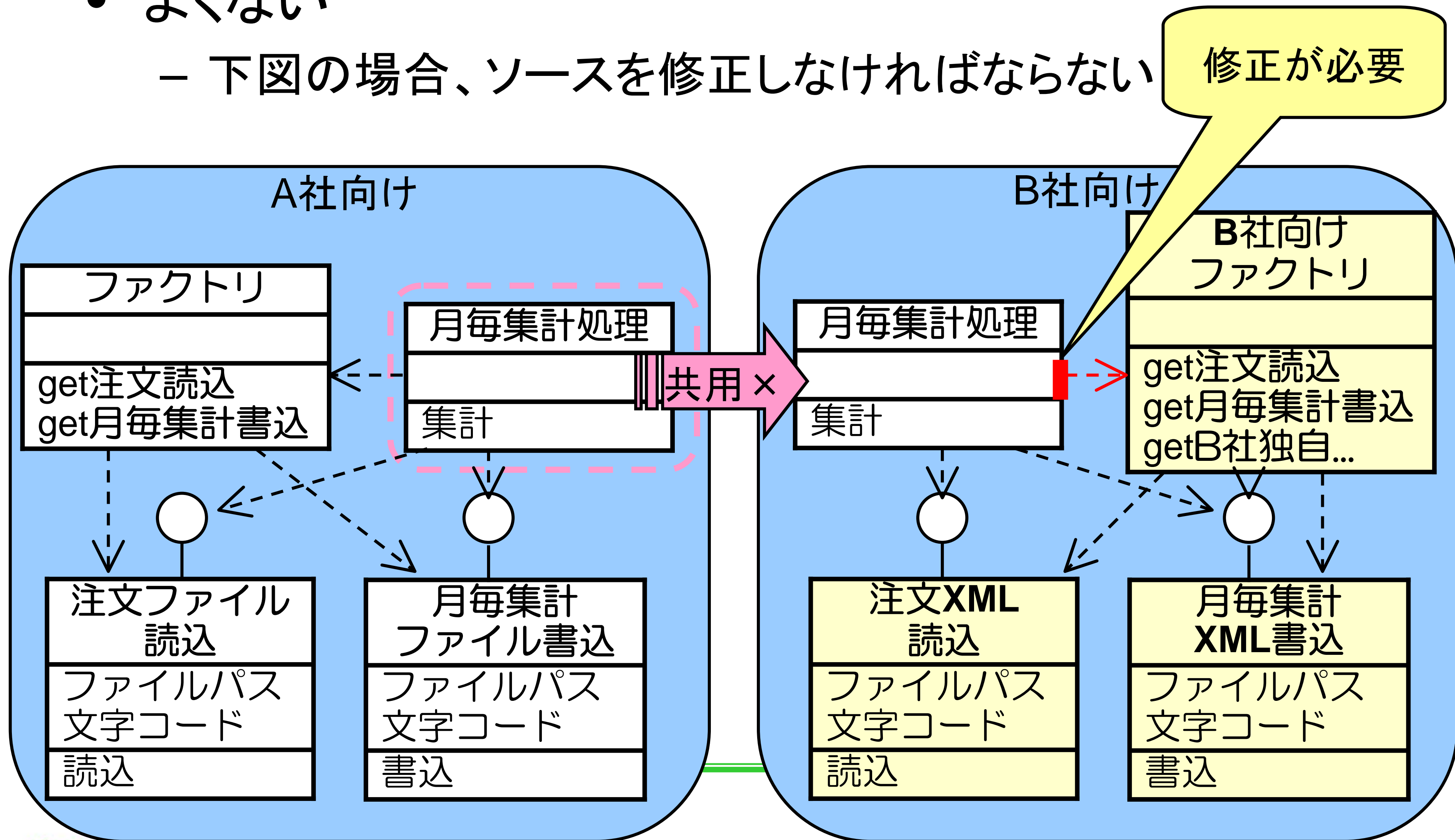
テスト容易性はどうか？

- ややよい
  - 月毎集計処理クラスを修正せずにモッククラスを使用できる(ファクトリクラスの修正は必要)



再利用性はどうか？

- よくない
  - 下図の場合、ソースを修正しなければならない

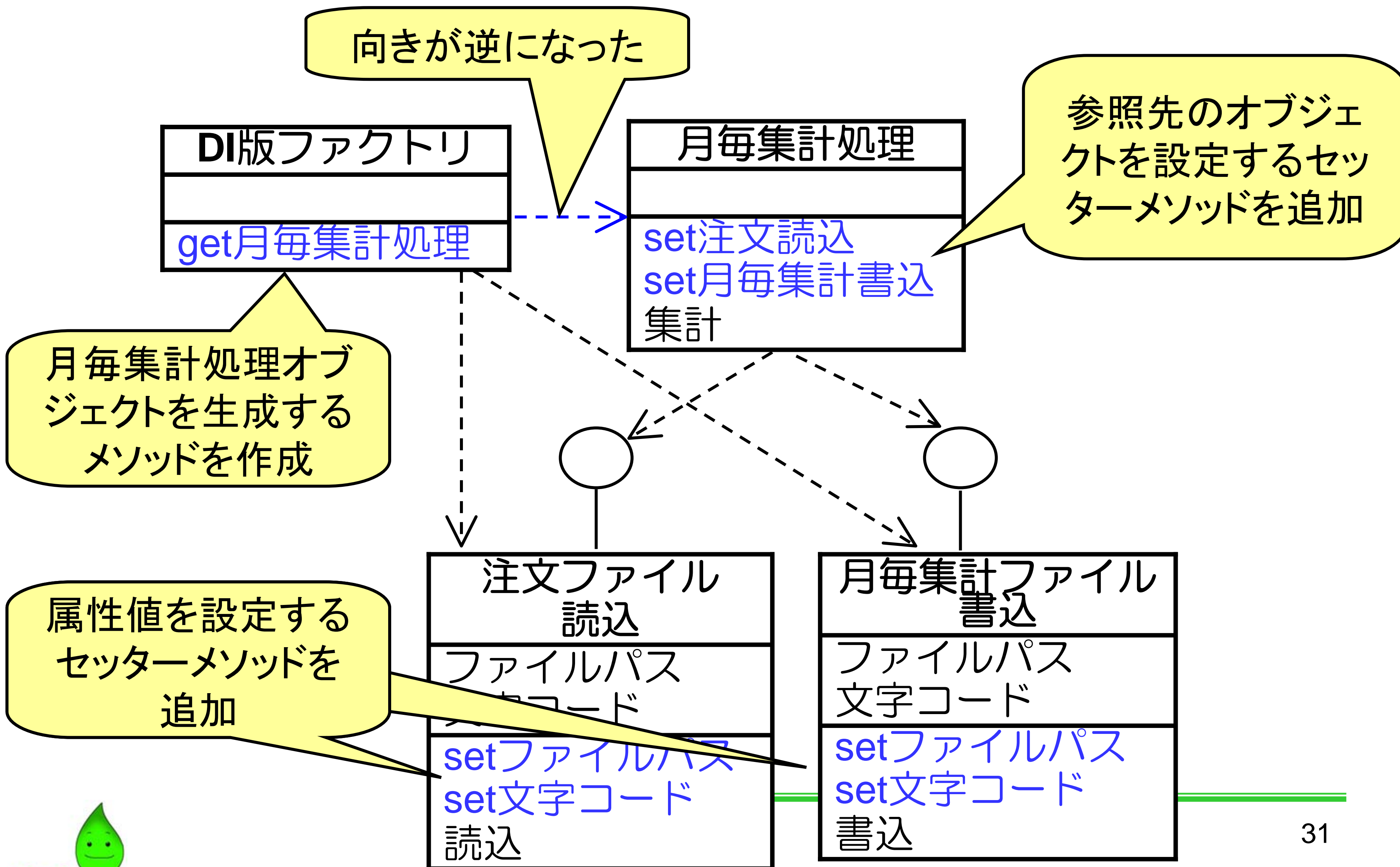


④DIを適用

- DIの思想
  - 設定と実装(本来の責務)の分離
    - 参照先のオブジェクト(プリミティブ型も含む)を自分で作らない・探さない
    - ハリウッドの法則
      - Don't call us. We'll call you

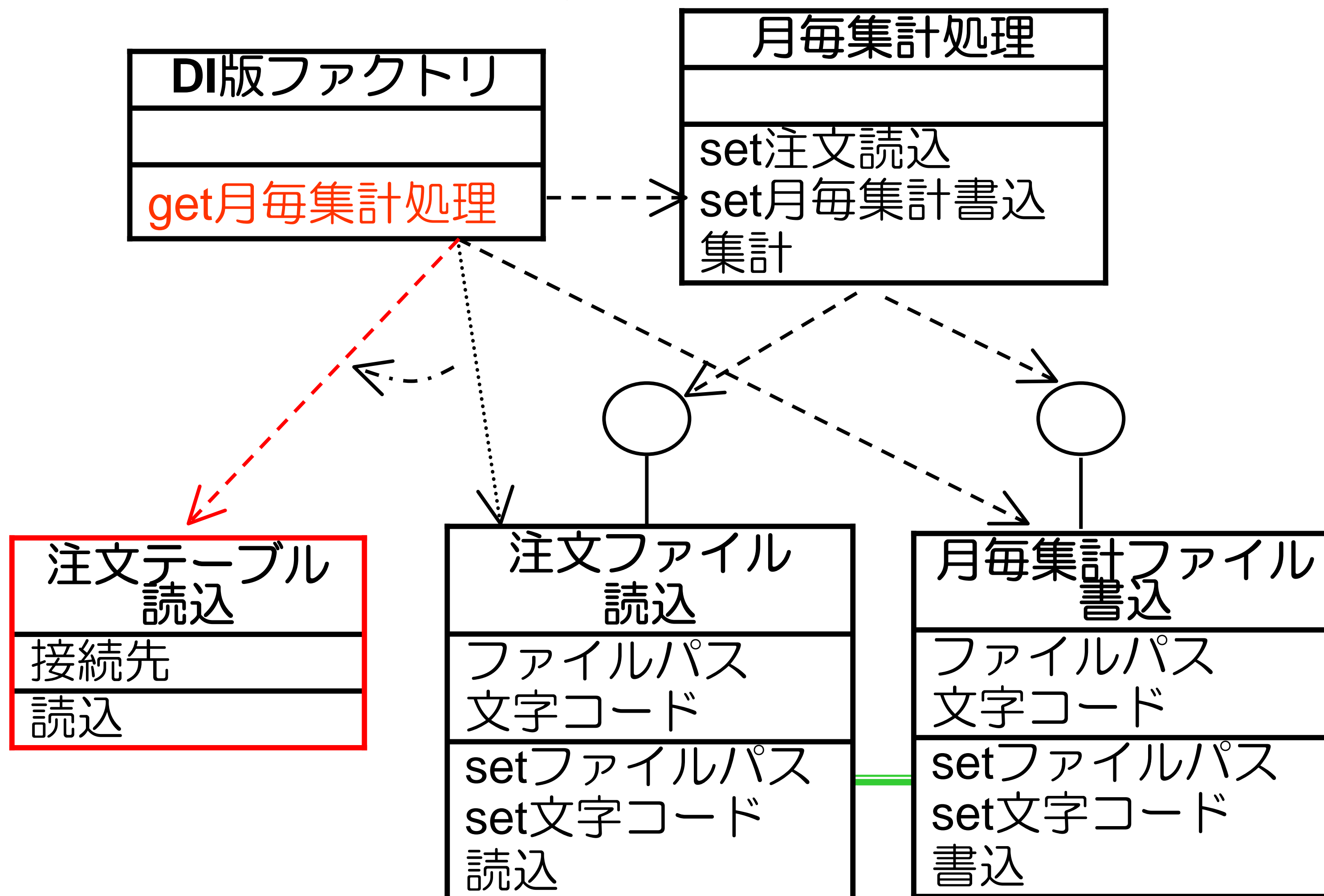
JSUG

DIを適用した場合



変更容易性はどうか？

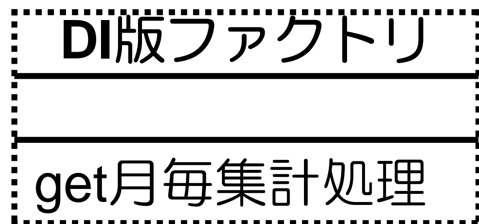
- よい
  - ファクトリクラスを修正するだけでよい

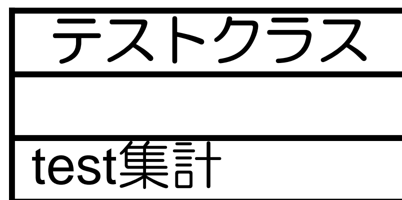


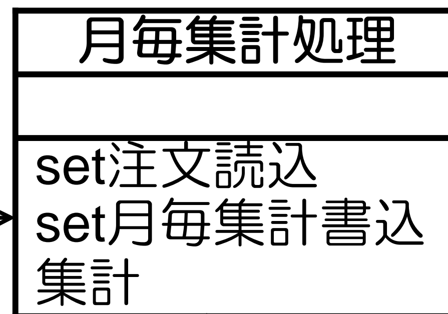
JSUG

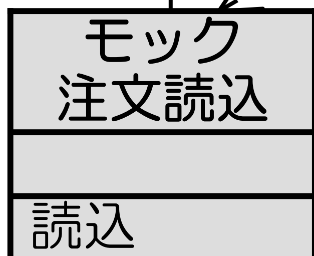
テスト容易性はどうか？

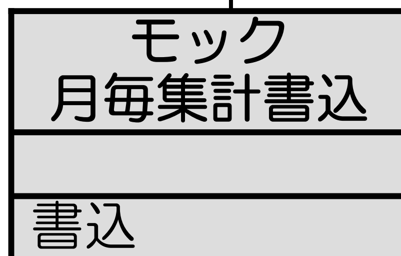
- よい
  - ファクトリを使わない

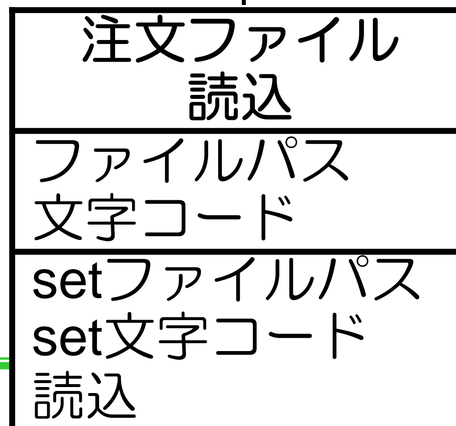


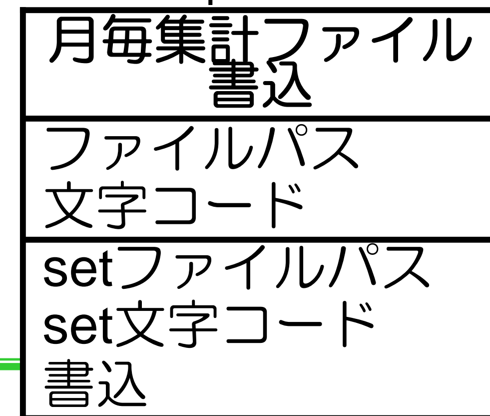






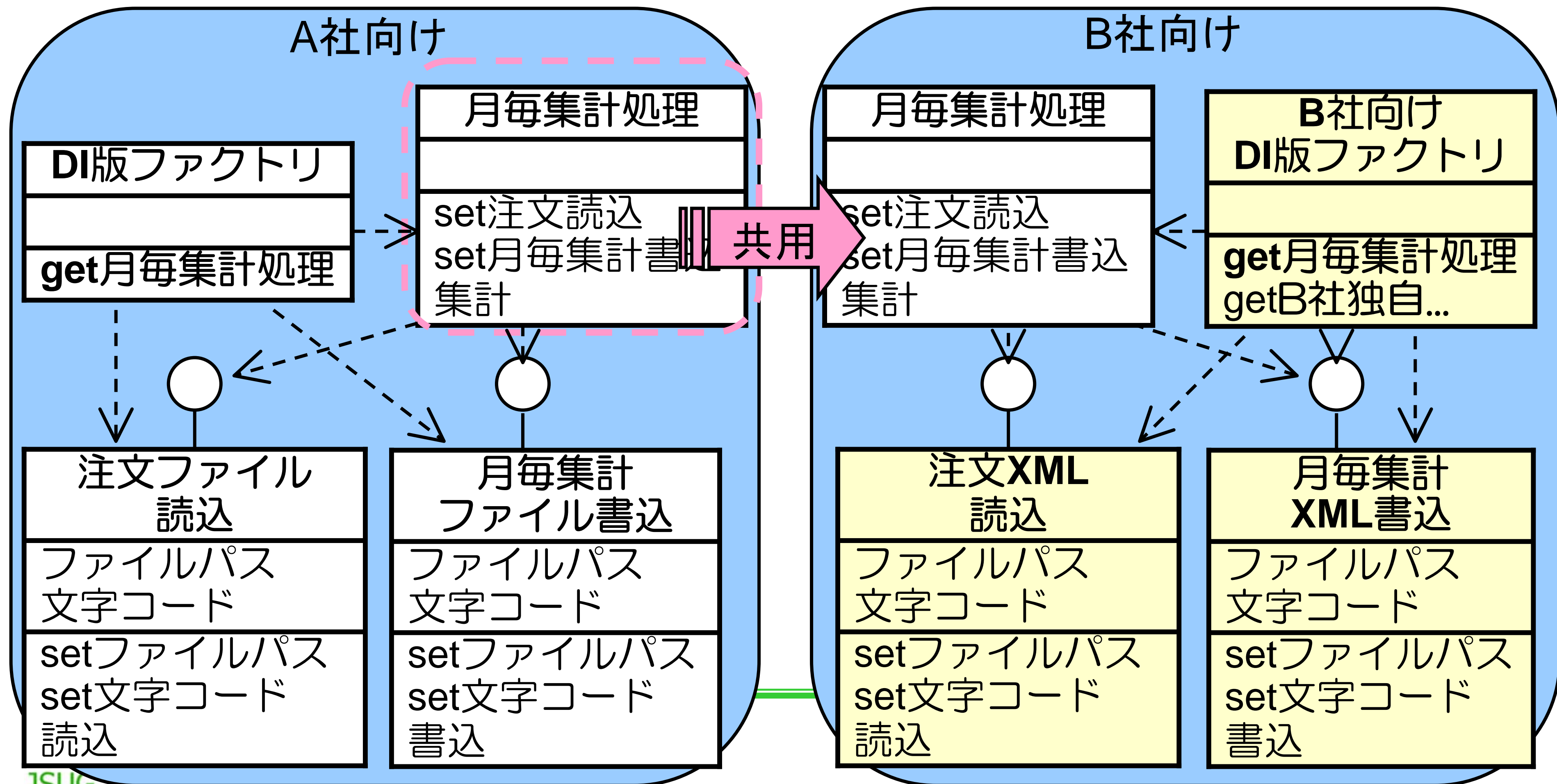






再利用性はどうか？

- よい
  - 月毎集計処理クラスを共用できる



設定と実装(本来の責務)の分離

- 開発者は、クラスの本来の責務を実装することに注力できる

参照するオブジェクトを探さないといけない・・・
集計処理も作らないといけない・・・

DIを適用

参照は用意されているもの。
集計処理だけ作ればよい。

集計クラス

集計クラス

DIのまとめ

- DI適用時に行うこと
  - オブジェクト間の関連の作成(プリミティブ型も含む)を、外部の誰かに任せる
    - 自分で作らない・探さない
    - 関連を注入してもらうための仕組みを用意する
- 効果
  - 変更容易性・テスト容易性・再利用性が向上
    - ソフトウェアの品質が向上

DIコンテナ

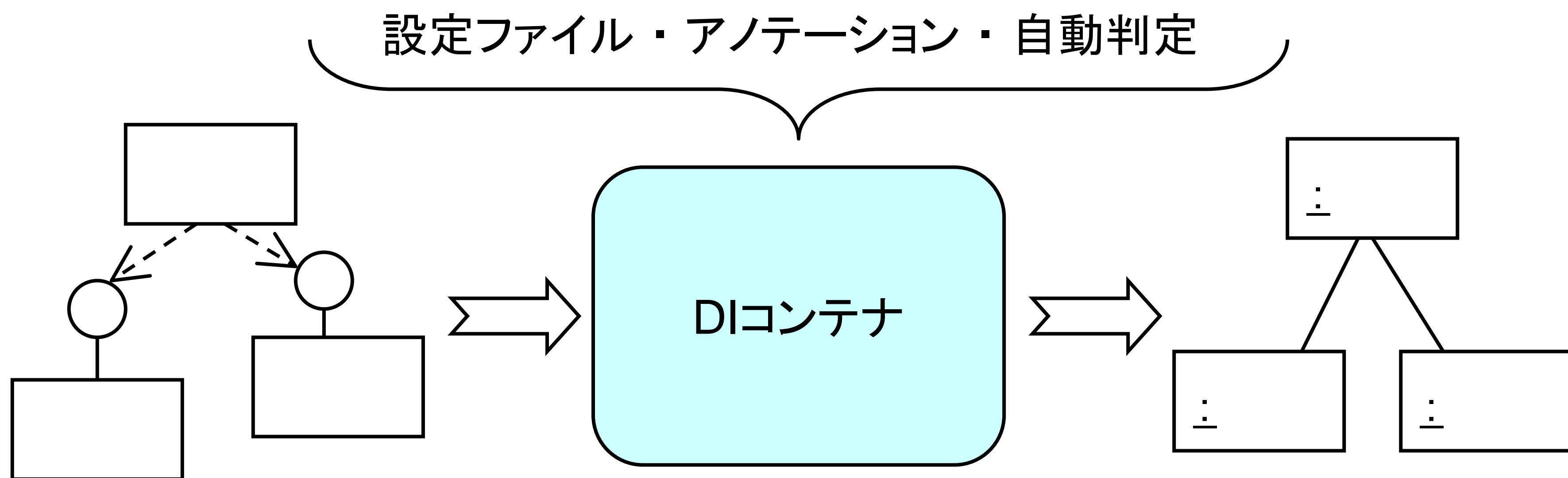
JSUG

DIコンテナとは？

- ソフトウェアの一種
- オブジェクトを組上げてくれる
  - オブジェクトを生成し、オブジェクト間の関連を作成する
- 組上げたオブジェクトを保管する入れ物
  - 組上げられたオブジェクトはDIコンテナに保管され、必要に応じて参照することができる
- さまざまな製品が存在する
  - Spring Framework、Seasar2、Guice、・・・

オブジェクトの組上げ

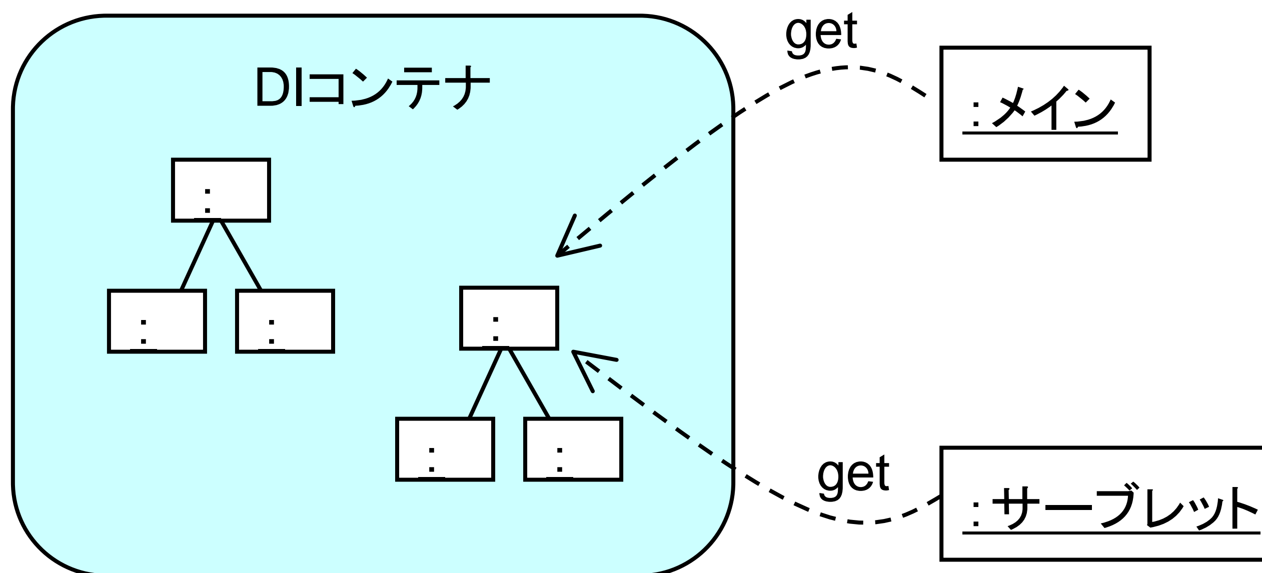
- 設定ファイル等の指示に従って、オブジェクトの生成やオブジェクト間の関連の作成を行う



JSUG

オブジェクトの入れ物

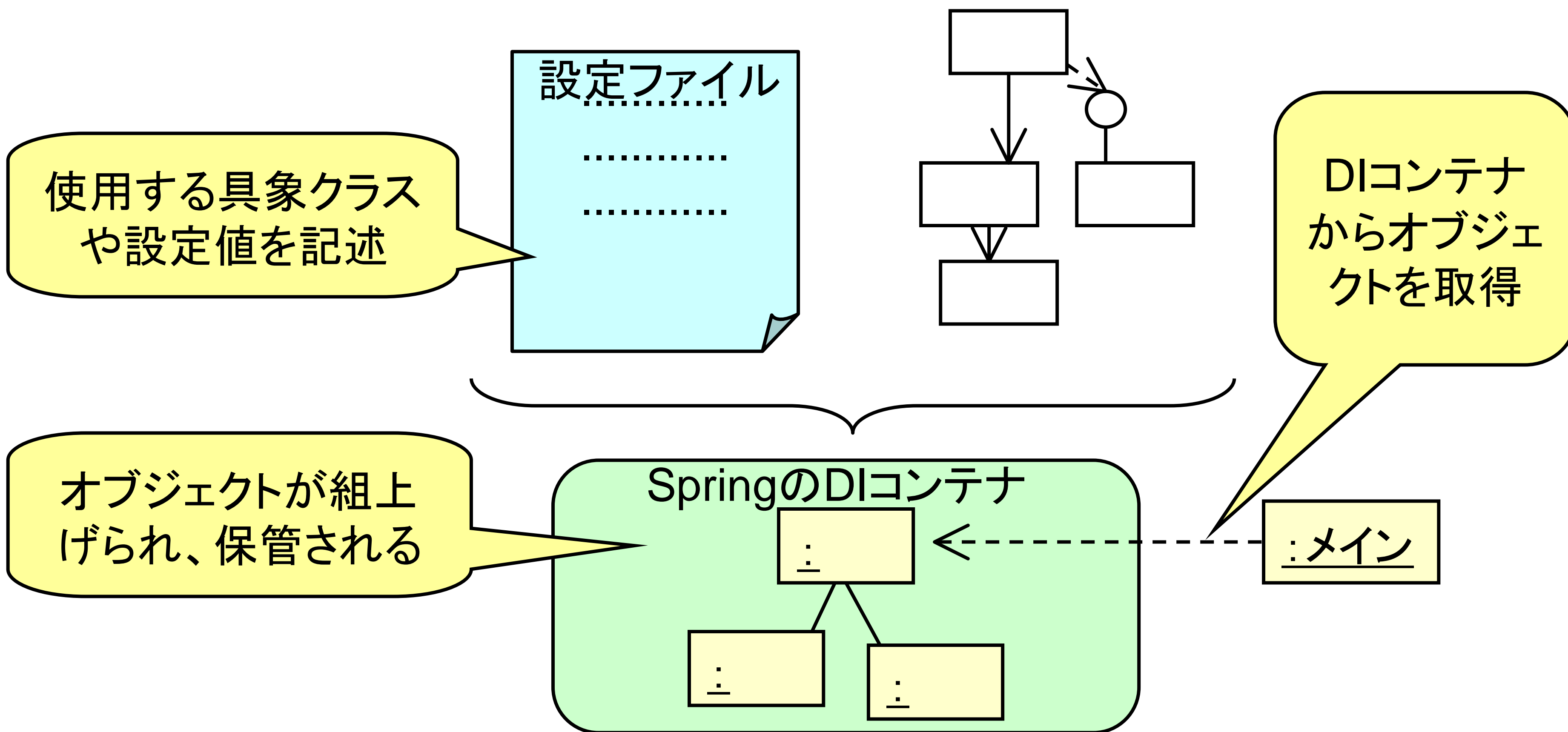
- 組上げられたオブジェクトはDIコンテナに保管され、必要に応じて参照することができる



Spring Framework

- DIコンテナの先駆け
- オープンソース
- 無償版と有償版がある
  - メンテナンスバージョンの提供方針やサポートに違いがある
- 世界的な知名度が高い

利用イメージ



DIの使用例

- 設定ファイル (sample/spring.xml)

```
<beans>
  <bean id="syukeisyori" class="sample.月毎集計処理">
    <property name="注文読込" ref="reader"/>
    <property name="月毎集計書込" ref="writer" />
  </bean>

  <bean id="reader" class="sample.注文ファイル読込">
    <property name="ファイルパス" value="入力ファイル/2008注文.tsv"/>
    <property name="文字コード" value="SJIS" />
  </bean>

  <bean id="writer" class="sample.月毎集計ファイル書込">
    <property name="ファイルパス" value="出力ファイル/2008月毎集計.tsv"/>
    <property name="文字コード" value="SJIS" />
  </bean>
</beans>
```

DIの使用例

- 実行

```
public static void main(String[] args) throws Exception {
        ApplicationContext DIコンテナ =
            new ClassPathXmlApplicationContext("sample/spring.xml");

        月毎集計処理 syukeisyori =
            (月毎集計処理)DIコンテナ.getBean("syukeisyori");

        syukeisyori.集計();
}
```

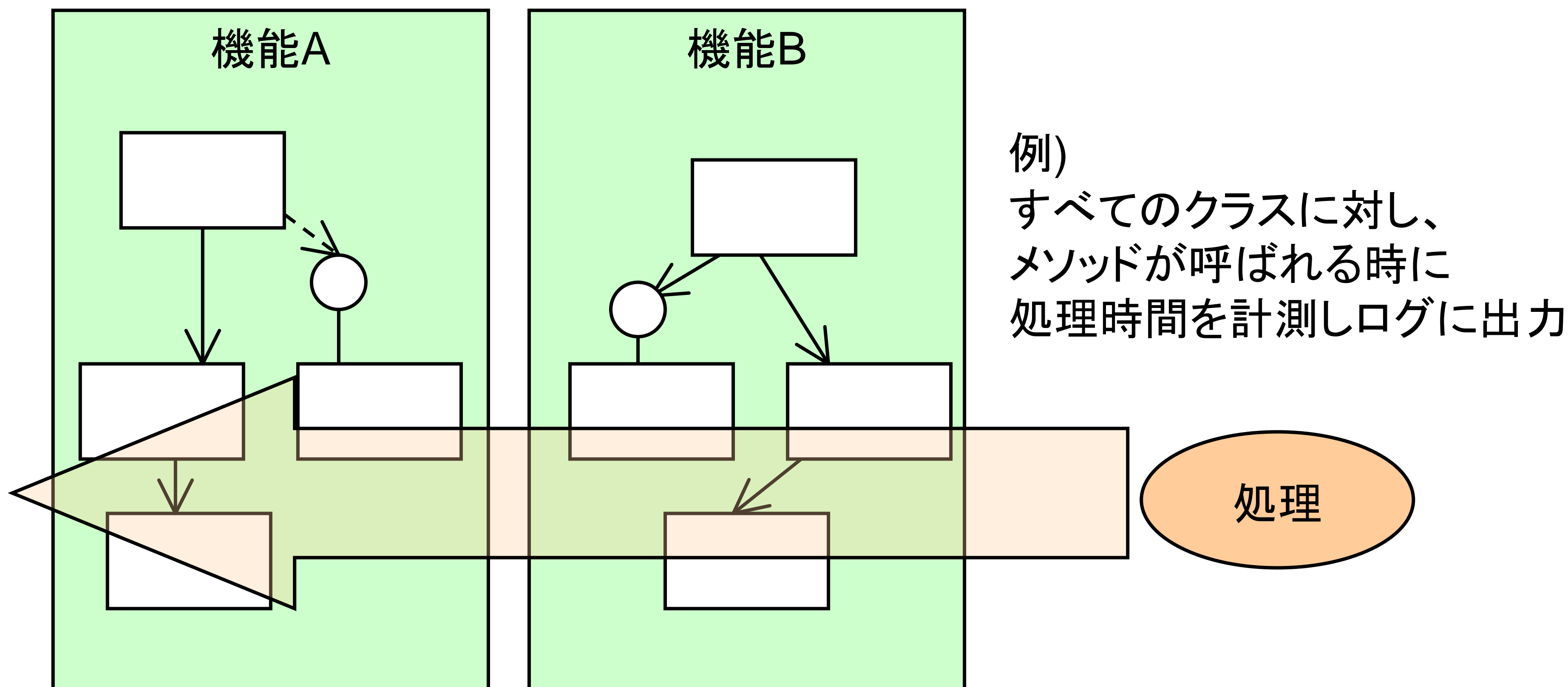
JSUG

AOP

JSUG

AOPとは？

- 横断的に処理を追加するプログラミングの手法
  - OOPを補完する



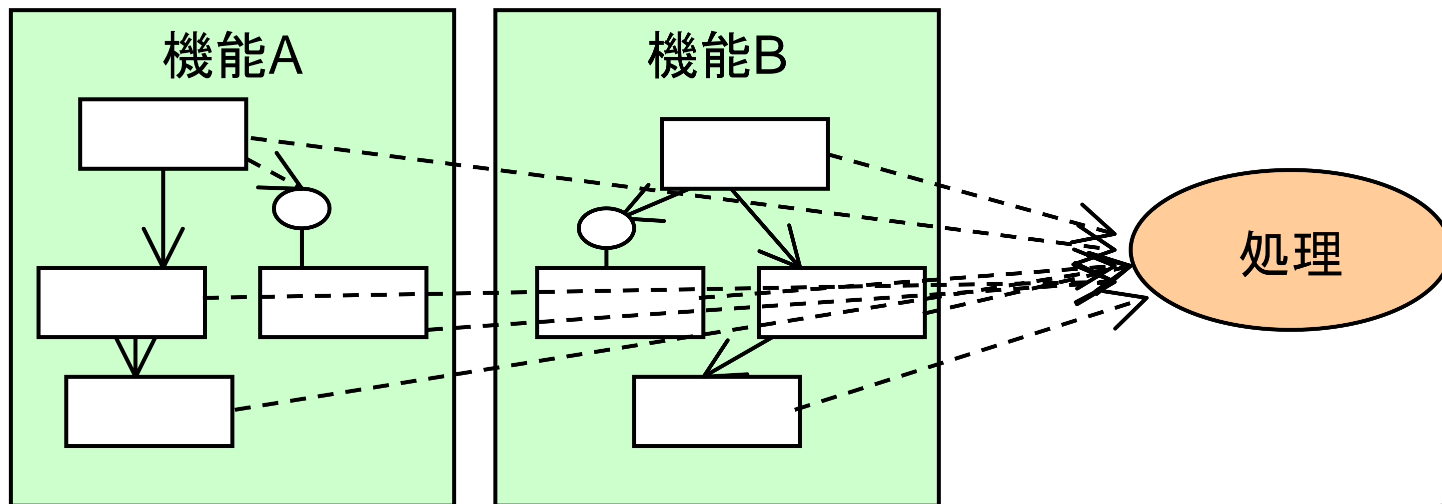
例)
すべてのクラスに対し、
メソッドが呼ばれる時に
処理時間を計測しログに出力

AOP(Aspect Oriented Programming：アスペクト指向プログラミング)
OOP(Object Oriented Programming：オブジェクト指向プログラミング)

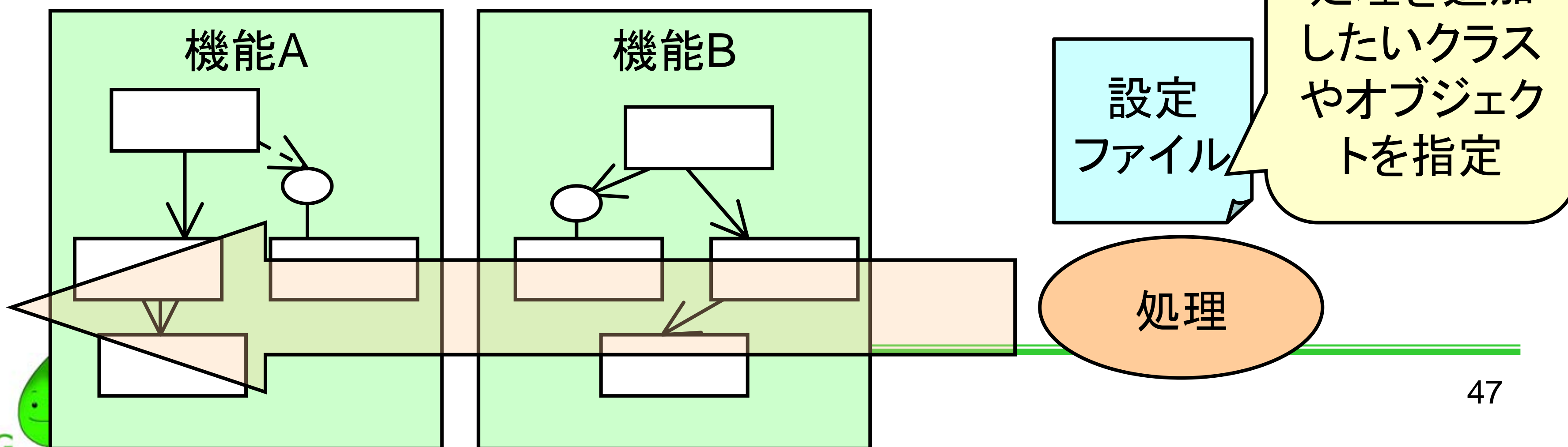
JSUG

AOPとは？

【非AOP】既存のモジュールを修正する必要がある



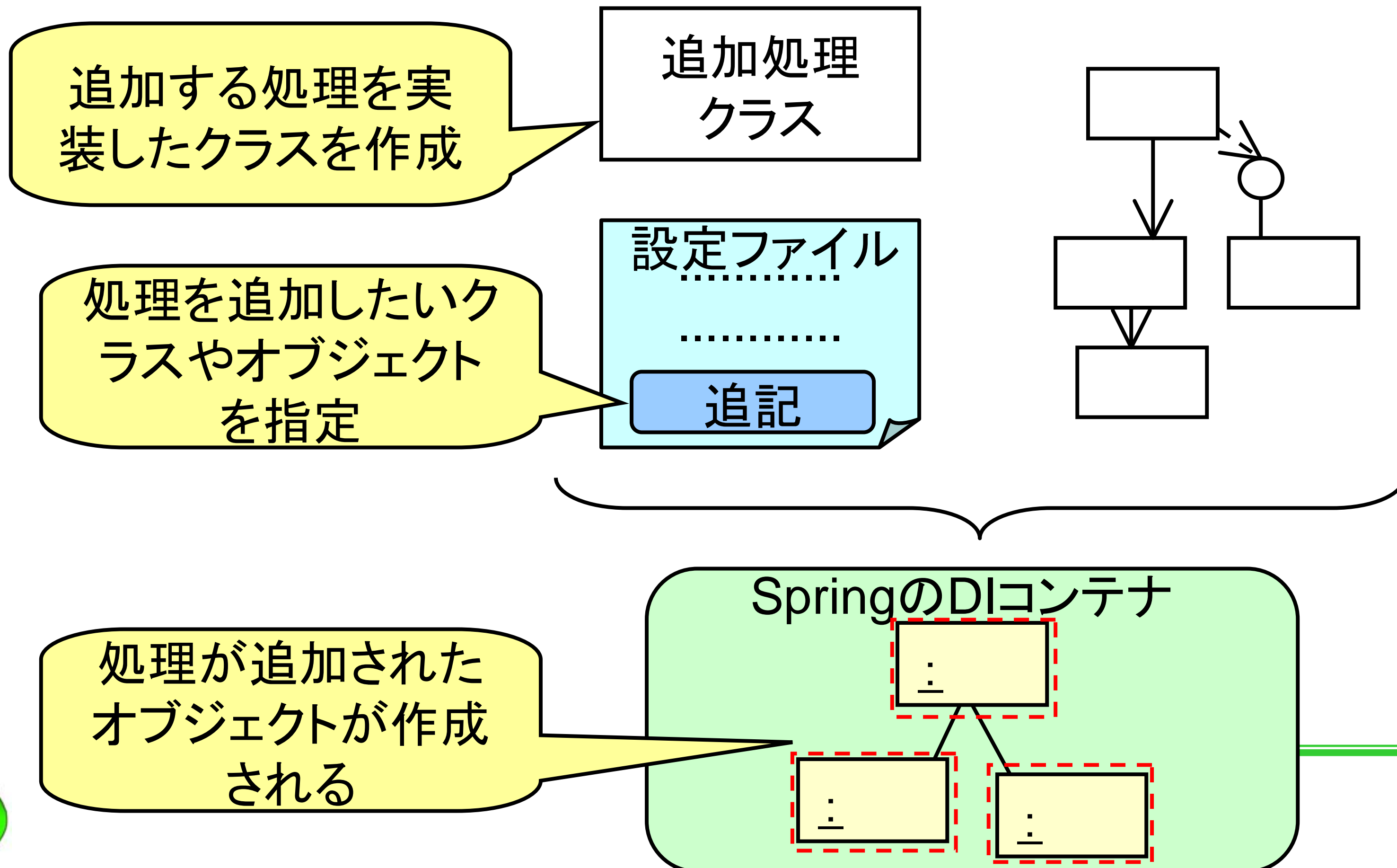
【AOP】既存のモジュールを修正せずに処理を追加できる



SpringとAOP

- SpringはAOPの機能を提供する
  - 他のDIコンテナ製品も大抵提供している

【利用イメージ】



AOPの利用例

- 追加したい処理の実装(メソッドの処理時間を計測)

```
public class 時間計測 implements MethodInterceptor {

    @Override
    public Object invoke(MethodInvocation v) throws Throwable {
        long start = System.currentTimeMillis();
        Object ret = v.proceed();
        long end = System.currentTimeMillis();
        System.out.println("メソッド名 : "+v.getMethod().getName()+
                " 処理時間(ミリ秒)は"+ (end - start) +"です。");
        return ret;
    }
}
```

JSUG

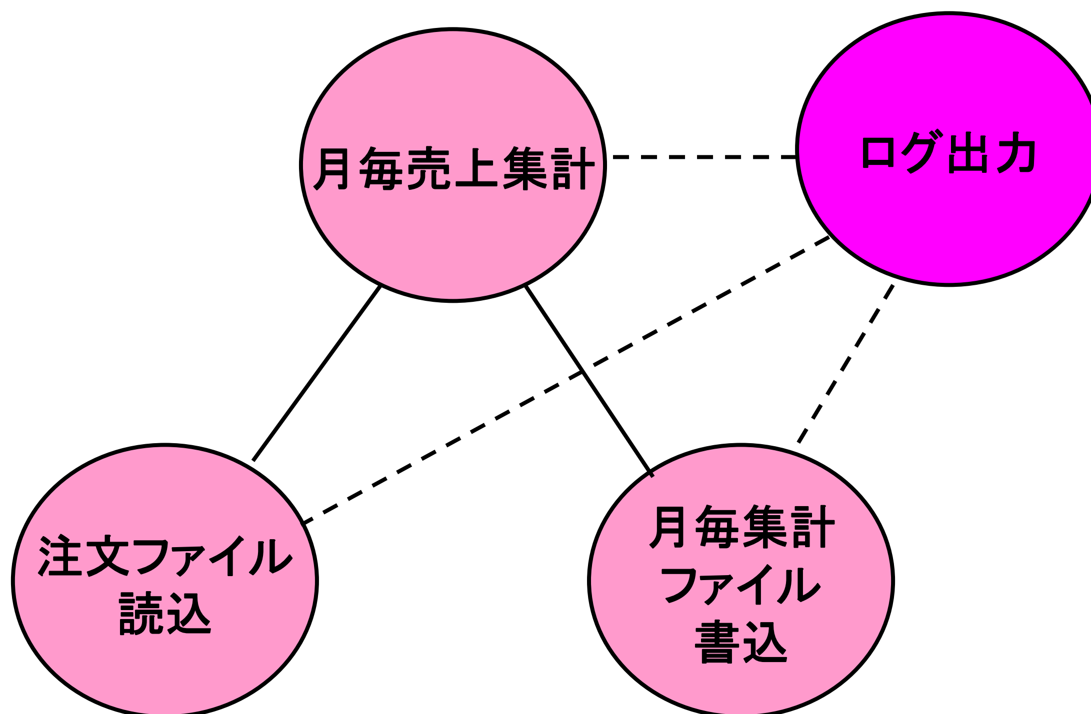
AOPの利用例

- 設定ファイル(sample/spring.xmlに追記)

```
<beans>

  ...

  <bean id="keisoku" class="sample.時間計測"/>
  <aop:config>
    <aop:advisor advice-ref="keisoku" pointcut="execution(* sample.*.*(..))"/>
  </aop:config>

</beans>
```

AOPとモジュール

- AOPで追加する処理もモジュールの位置づけ



JSUG

質疑応答

JSUG

JSUGについて

- Japan Spring User Group
  日本Springユーザ会

- メンバー数：346名(2008/08/08現在)
  – 誰でも無料で参加できます

- ２ヶ月に１回程度の勉強会
  – 勉強会で使用した資料は以下のURLにあります
    - http://groups.google.co.jp/group/jsug/files

JSUG

ご清聴ありがとうございました

JSUG

ライセンスについて

JSUGマスコットアイコン（本スライド左下）が残されている場合に限り、本作品（またそれを元にした派生作品）の複製・頒布・表示・上演を認めます。

非商用目的に限り、本作品（またそれを元にした派生作品）の複製・頒布・表示・上演を認めます。

本作品のライセンスを遵守する限り、派生作品を頒布することを許可します。

JSUG