Agility for Business
Technologic Arts

第4回Java超入門
~デザインパターン基礎~
WEBプログラマーへの第一歩

テクノロジックアート
長瀬　嘉秀

今日の目標

Agility for Business
Technologic Arts

- デザインパターンの前提となる基礎知識を習得する
- デザインパターンの位置付けを理解する
- デザインパターンの適用目的を理解する
- GoFのデザインパターンの概要を理解する
- パターン内のクラスやインスタンスの協調関係を理解する

今日の流れ

Agility for Business
Technologic Arts

- デザインパターンの前提となる基礎知識を説明
- デザインパターンの位置付けを説明
- デザインパターンの詳細を説明
- まとめ

目次

Agility for Business
Technologic Arts

勉強会の参考書

Agility for Business
Technologic Arts

独習デザインパターン
【発売日】2004年1月24日
【著作】株式会社テクノロジックアート
【監修】長瀬 嘉秀
【出版】株式会社翔泳社
【ISBN】4-79810-445-0

Agility for Business
Technologic Arts

第１章　オブジェクト指向の基礎

デザインパターンとは

Agility for Business
Technologic Arts

- 設計で頻出する問題に対する効果的な解法
- 解法が次のように体系化され、まとめられている
  - どのような問題点を克服するためのものか、
  - どのようなときに適用できるのか、
  - どのような利点が得られるのか、
  - どのようなことに注意しなければならないのか
- 洗練され、再利用性、柔軟性、拡張性を高めるための
  ノウハウが詰め込まれている

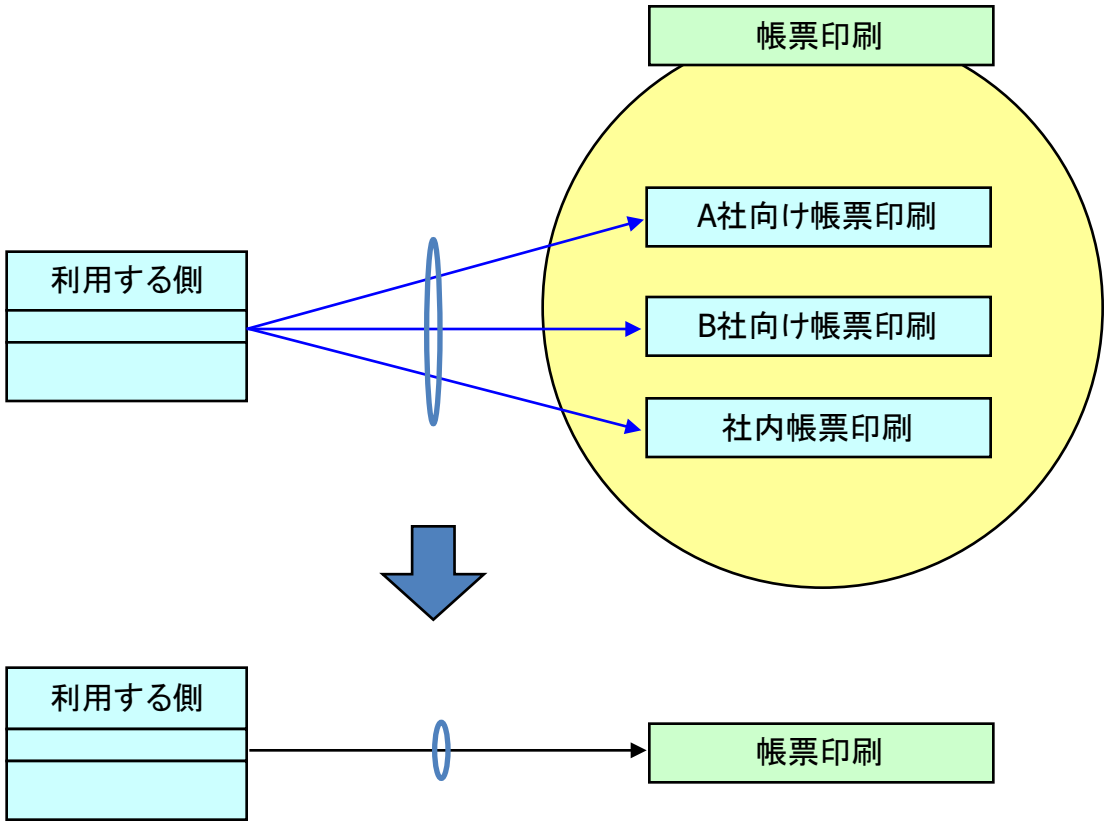
デザインパターンの学習前に…

Agility for Business
Technologic Arts

- オブジェクト指向の基礎知識
  - 抽象化
  - ポリモーフィズム
  - 粒度/カプセル化
  - 可視性
- オブジェクト間の関連
  - ホワイトボックス
  - ブラックボックス
- パターンについて

抽象化(概念図)

Agility for Business
Technologic Arts



抽象化(解説)

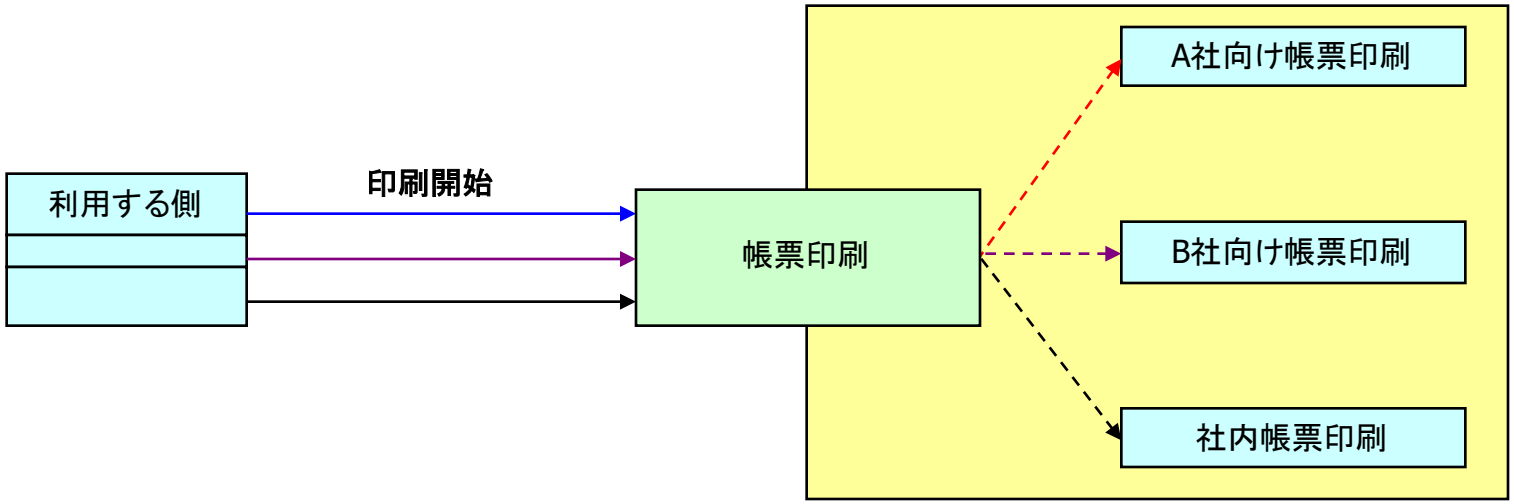
Agility for Business
Technologic Arts

- **オブジェクトを統一の概念でまとめる**

- オブジェクトを概念レベルでグループ化するため、一つのまとまりとして理解することができる
- オブジェクトの詳細を一時的に頭の外に追いやることができる
  - → 概念を共有することで、グループ内のオブジェクトが果たすべき責任(仕様)を理解しやすくなる
  - → 概念を共有することで、グループ内のオブジェクトの振る舞いが連想しやすくなる
  - → 詳細は必要に応じて知ればよくなる

ポリモーフィズム（概念図）

Agility for Business
Technologic Arts



ポリモーフィズム(解説)

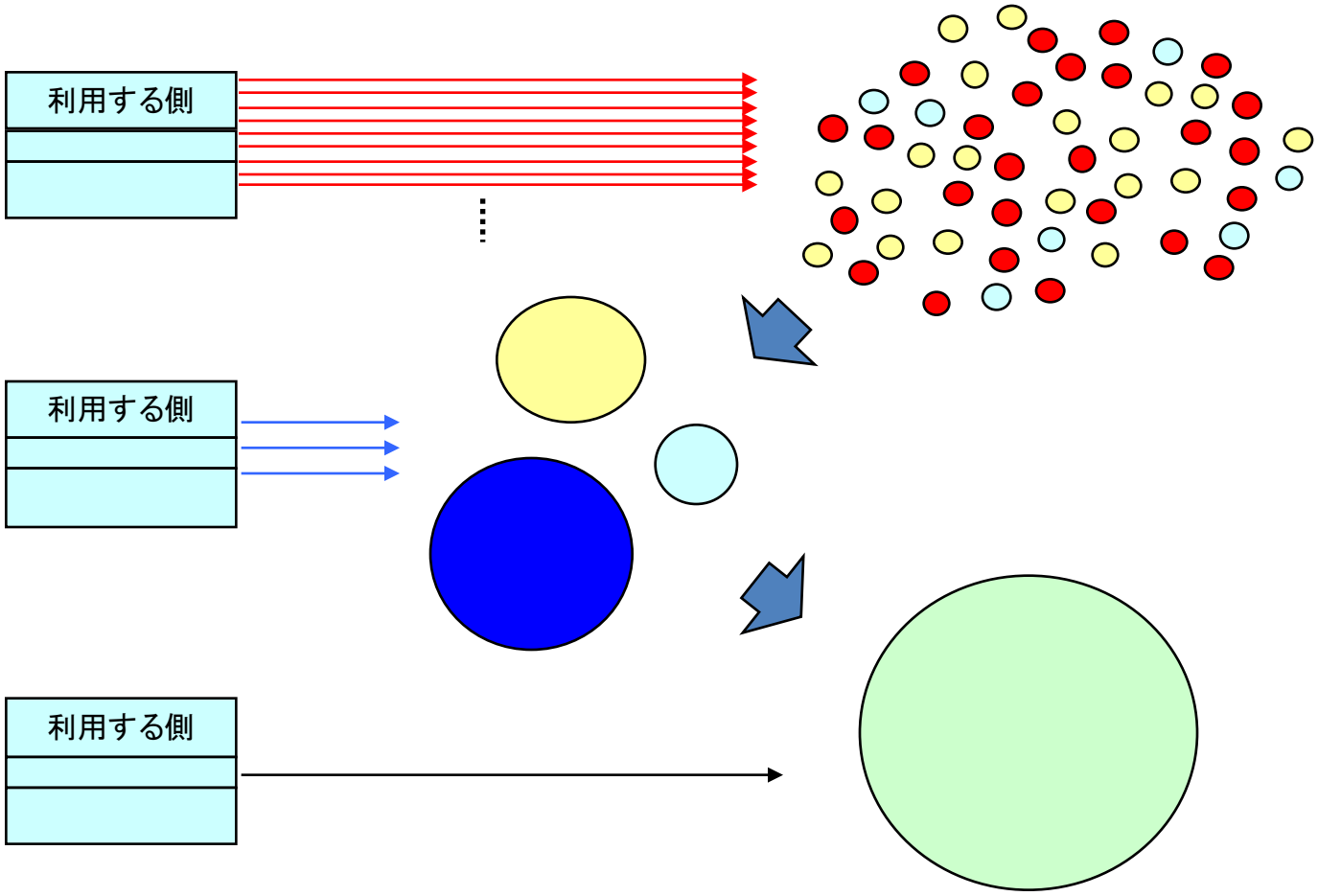
Agility for Business
Technologic Arts

- **オブジェクトを統一のインターフェースで扱う**
- **オブジェクトの型からの独立を支援する**

- 仕様追加の影響を最小限に押さえられる
  → 利用者は抽象オブジェクトを扱うため、グループに新たなオブジェクトが追加されたことを意識しなくて良くなる
- 複数の異なるオブジェクトを同一のメッセージで扱える
  → オブジェクトの扱いが抽象化により統一される
  具体的なオブジェクトの型を知らなくて済む
  → グループに新たなオブジェクトが追加されたことを
  気にしなくて良くなる
  インターフェースが重要であって、実装は二の次とする

粒度/カプセル化(概念図)

Agility for Business
Technologic Arts



粒度/カプセル化(解説)

Agility for Business
Technologic Arts

- **適切な単位でグループ化を行う**
  関数・データ ⇒ クラス ⇒ コンポーネント ⇒ サブシステム

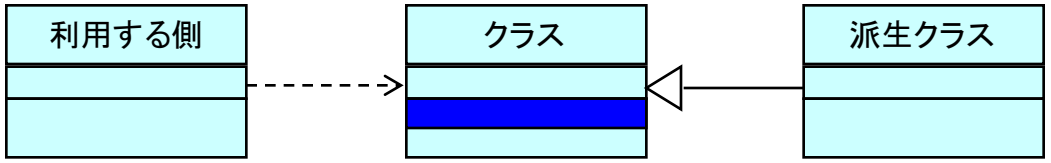
- 扱いやすく、関連のあるグループでまとめる
  → 適切にグループ化されていると、グループ内のものの関連性が理解しやすい
  → 扱いやすいインターフェースを提供することで、グループ内の詳細を知らなくてもよくなる
  → 情報隠蔽(カプセル化)を行える

オブジェクト間の関連（概念図）

Agility for Business
Technologic Arts

- クラス継承（ホワイトボックス再利用）

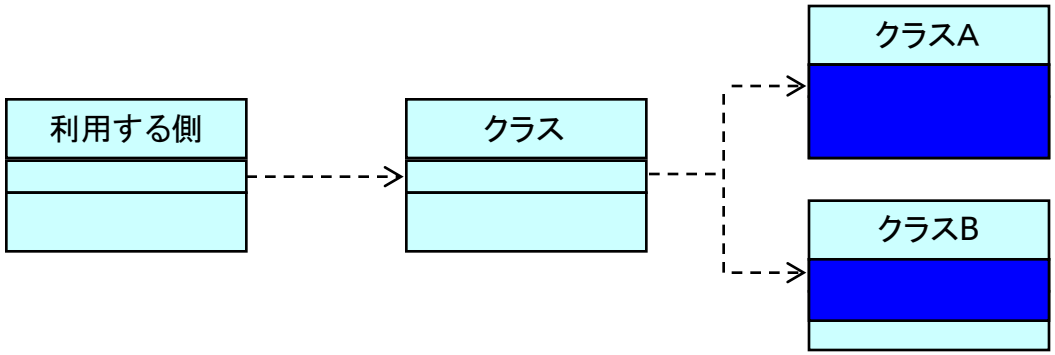
スーパークラス利用



再利用箇所

- オブジェクトコンポジション（ブラックボックス再利用）

インターフェース利用



オブジェクト間の関連(解説)

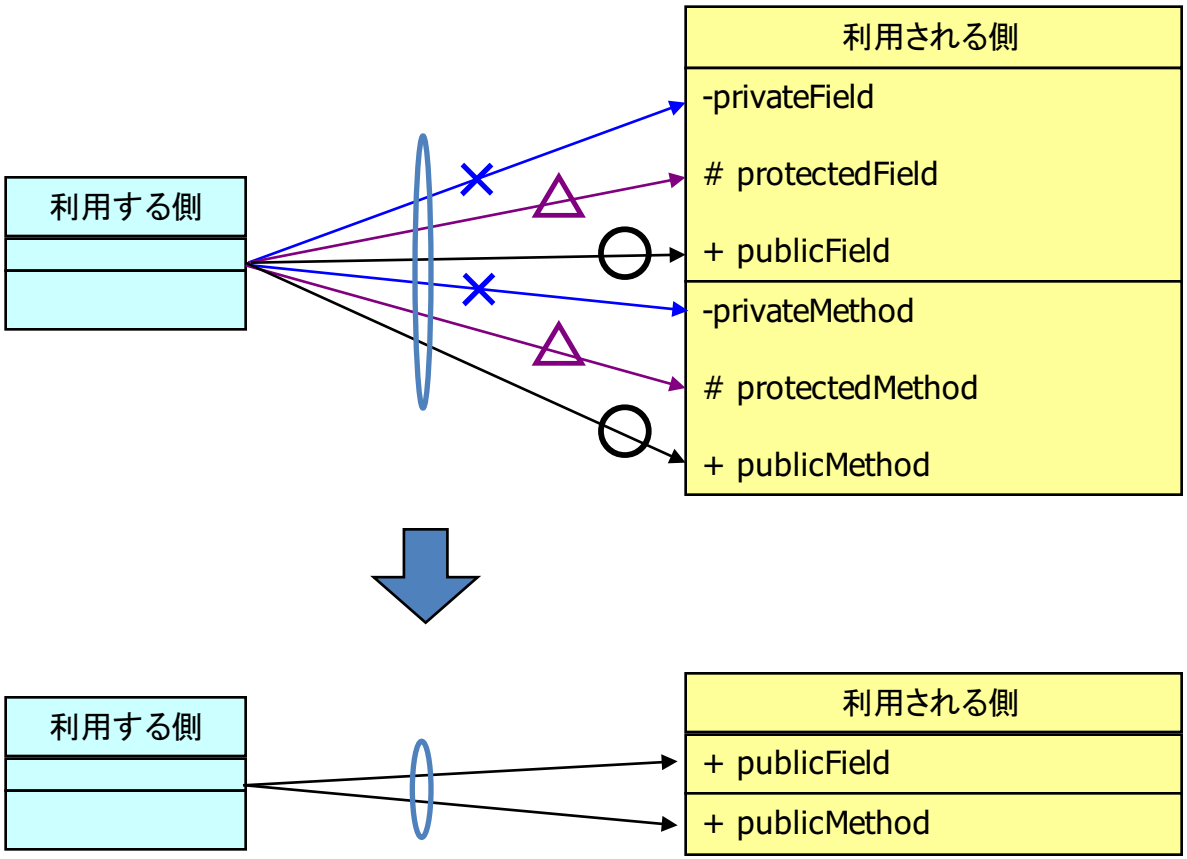
Agility for Business
Technologic Arts

- **再利用の目的、拡張性などを考慮して関連を選択する**

- **クラス継承 ： 強い関連性**
  → 概念の共有して、機能を拡張できる
  → 再利用した実装を必要に応じて拡張し、利用できる
- **オブジェクトコンポジション ： 弱い関連性**
  → 概念を借りて、新たな概念・機能を生み出せる
  → 再利用した実装をそのまま利用する
  → クラス継承の数を減らすことができる
  → 実行時にオブジェクトの組換えができる

可視性(概念図)

Agility for Business
Technologic Arts



可視性(解説)

Agility for Business
Technologic Arts

- **オブジェクトへのアクセスを保護する**
- **インターフェースを明確にする**

- オブジェクトの内部データの扱い方を限定できる
  → 内部データの一貫性を保証できる
- オブジェクトの内部データの構造の自由度が上がる
  → 内部データを扱う責任はオブジェクトが持つため、
  インターフェース仕様に沿えば、データの構造は限定されない
- オブジェクトの詳細を知る必要がない
  → オブジェクトがどのようなの責任を持ち、どのような
  インターフェースで責任を果たすのかを知るだけでよい

設計意図の反映

Agility for Business
Technologic Arts

- Java言語では以下の方法で、ソースコードに設計意図を反映できます

- final ： 継承の抑制、オーバーライドの抑制を行う
  → 派生クラスで一貫性が損なわれるのを防ぐ
- abstract ： 抽象クラス、抽象メソッドの宣言を行う
  → 派生クラスの作成が前提となっていることを表す
  また、抽象メソッドとすることで実装を義務付ける
- interface : インターフェースの宣言を行う
  → 全てのインターフェースの実装を義務付ける
- 可視性（private, protected, public）：使用制限を明確にする
  → インターフェースを明確にする

ここまでのまとめ

Agility for Business
Technologic Arts

- 一貫性の確保と概念の共有
- 抽象化とポリモーフィズム
- インターフェースの明確化
- 粒度の適正化

- いかに分かりやすくするか
- いかに依存関係を減らすか
- いかに柔軟性・拡張性を高めるか
- いかに再利用性を高めるか

Agility for Business
Technologic Arts

第2章　デザインパターン概要

パターンとは

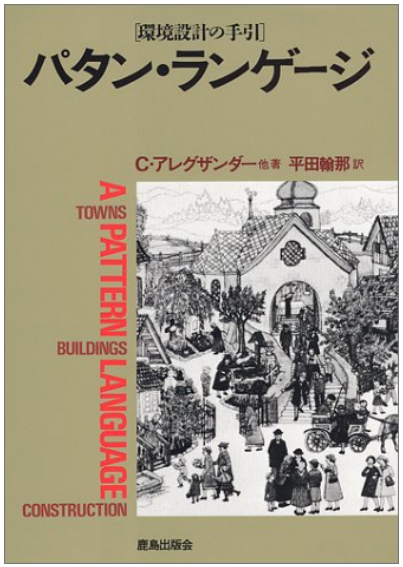
Agility for Business
Technologic Arts

- 頻繁に発生する問題に対する、
  効果的な解法の型
- 完全に同じ問題ではなく、
  ある状況下の問題を対象とする
- 過去に何度も使われている解法
- ソフトウェアパターン

パターンの歴史

Agility for Business
Technologic Arts

- 建築分野でのパターン
- 『 A Pattern Language 』
- Christopher Alexander
- 邦訳 : パタン・ランゲージ、平田 翰那 (翻訳)、鹿島出版会



パターンの歴史

Agility for Business
Technologic Arts

- 建築のパターンをソフトウェアに適用
- デザインパターン
- 数々のカンファレンス

  TOOLS, ECOOP, OOPSLA
- 現在のパターンコミュニティ

  PLoP, OOPSLA, JPLoP, etc.

パターンの分類

Agility for Business
Technologic Arts

- アーキテクチャパターン

  ソフトウエアシステムの構造と役割を定義
- アナリシスパターン

  分析時に適用するパターン
- デザインパターン

  設計時に適用するパターン
- イディオム

  プログラミング言語に特化したパターン

その他のパターン

Agility for Business
Technologic Arts

- アンチパターン

  やってはいけないことのパターン
- プロセスパターン

  開発プロセスのパターン
- 組織パターン

  人員管理に関するパターン
- J2EEパターン

  Java2 Enterprise Editionにおける設計のパターン

Agility for Business
Technologic Arts

デザインパターンとは

デザインパターンの目的

Agility for Business
Technologic Arts

- 洗練された設計を習得・再利用する
- ソフトウェアに安定した構造を導入する
- コードにたいする共通の理解を得やすくする
- コミュニケーションのための語彙とする

ソフトウェアの
再利用性、柔軟性、拡張性、保守性
を向上させる

デザインパターンの作成

Agility for Business
Technologic Arts

- 何度も同じような設計・実装をしてきた

- ソフトウェアの再利用性や柔軟性を高めたい

- 経験から獲得した解法の工夫を抽出・洗練する

- 問題ごとに解法の指針をカタログ化
  「パターン名、問題、解法、結果」

デザインパターンの意義

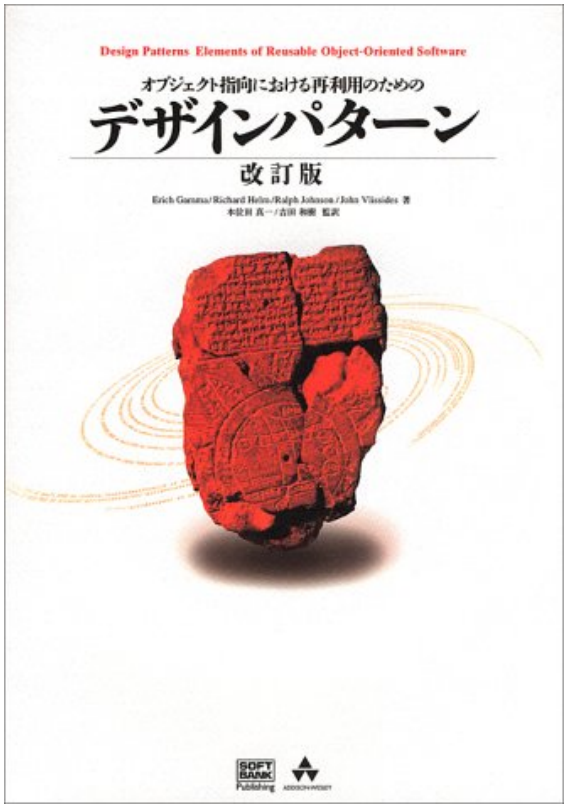
Agility for Business
Technologic Arts

- 適用範囲が明確に定義されている
  → 「課題の解法」を習得・再利用できる
- 洗練され、高い再利用性・保守性を持っている
  → 構造が安定し、再利用性を確保できる
- コミュニケーションのための語彙になる
  → 設計に対し、共通の認識を得ることができる
  → 開発時の前提知識の底上げができる
- モデルの知識は変化しにくい
  → 設計は実装言語に依存せずに再利用できる

GoFのデザインパターン

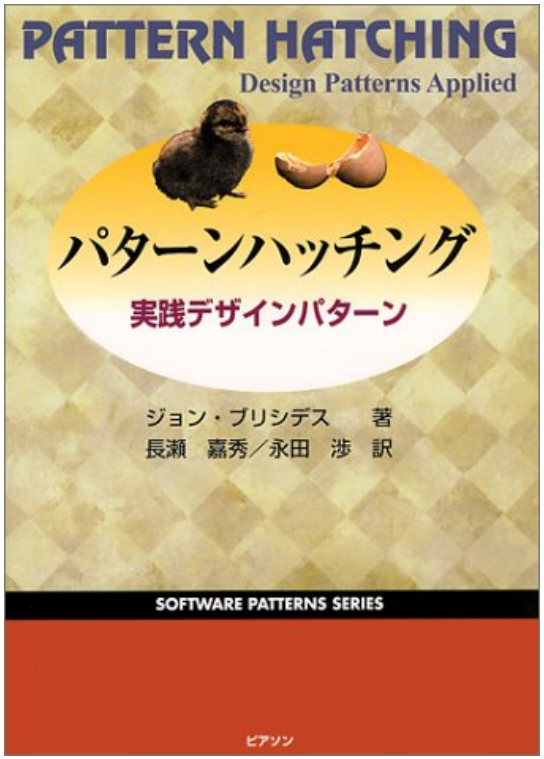
Agility for Business
Technologic Arts

- 「オブジェクト指向における再利用のためのデザインパターン 」
- ガンマ、ブリシデス、ヘルム、ジョンソン (著)
- 本位田真一、吉田和樹（監訳）、ソフトバンクパブリッシング



GoFのデザインパターン(2)

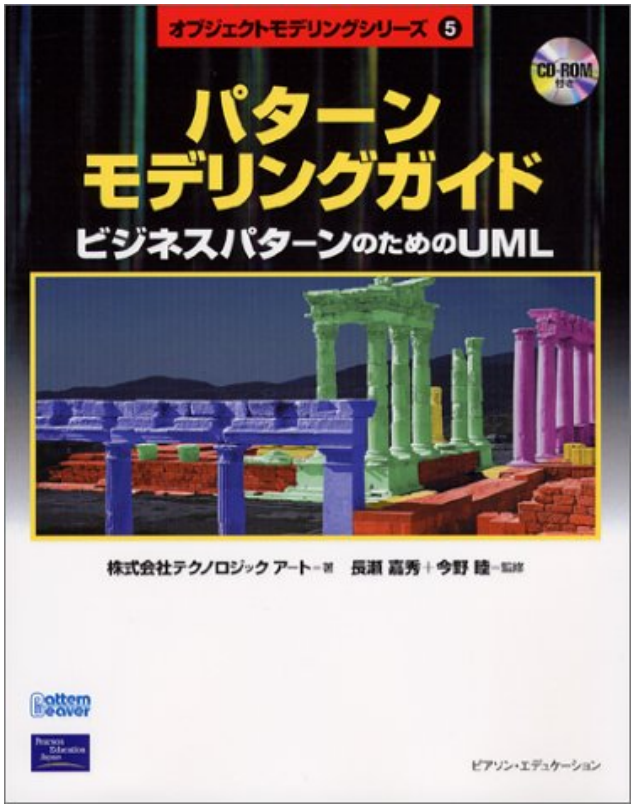
- 「パターンハッチング―実践デザインパターン 」
- ジョン ブリシデス (著)
- 永田渉、長瀬嘉秀(訳)、ピアソン・エデュケーション



GoFのデザインパターン(3)

Agility for Business
Technologic Arts

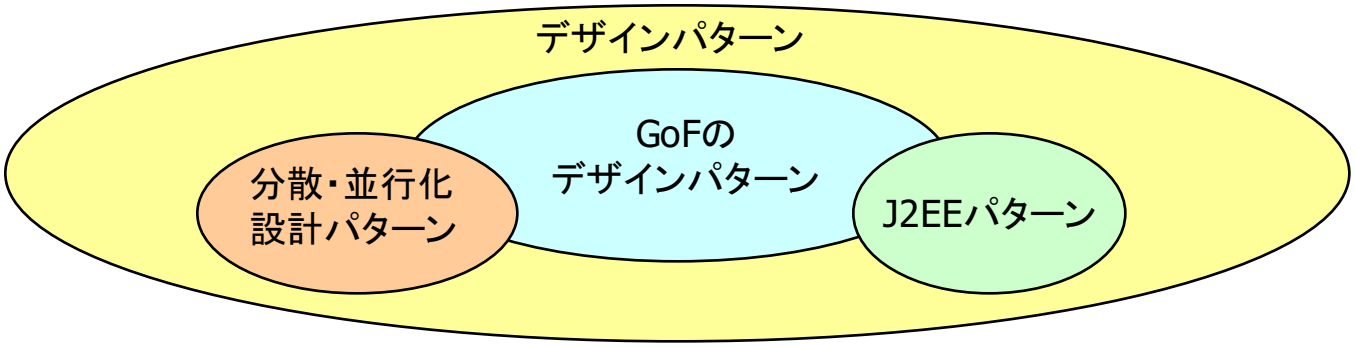
- 「パターンモデリングガイド」
- テクノロジックアート(著)
- 今野睦、長瀬嘉秀(監修)、ピアソン・エデュケーション



デザインパターンの分類

Agility for Business
Technologic Arts

- デザインパターンは設計上のパターン
  - GoFのデザインパターン（23パターン）
    GoF（Gang of Four）と呼ばれる四人がカタログ化
    : Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - J2EEパターン（15パターン）
    Sun Java Centerで、J2EEに有用なものをカタログ化
  - 分散・並行化設計パターン
    Pattern Languages of Programsワークショップ選出のものなど



生成に関するパターン

Agility for Business
Technologic Arts

- オブジェクトの生成プロセスに関する(全5パターン)

  オブジェクトがどのように生成、結合、表現されているのか、ということからシステムを独立にする方法を提供する

No.	パターン名	目的
1	Abstract Factory	オブジェクト群を明確にせず生成するためのインターフェースを提供する
2	Builder	複合的に組み合わせたオブジェクトを生成する
3	Factory Method	インスタンス化をサブクラスに任せる
4	Prototype	コピーして新しいオブジェクトを生成する
5	Singleton	インスタンスが1つしか存在しないことを保証する

解説対象のデザインパターン

構造に関するパターン

Agility for Business
Technologic Arts

- クラス、オブジェクトの構造に関する(全7パターン)
  クラスやオブジェクトを柔軟に合成する方法を提供する

No.	パターン名	目的
1	Adapter	インターフェースに互換性のないクラス同士を組み合わせる
2	Bridge	機能と実装を別々の階層で拡張する
3	Composite	オブジェクトを木構造に組み立てる
4	Decorator	動的にオブジェクトに責任を追加できるようにする
5	Façade	複数のインターフェースに高レベルの統一インターフェースを与える
6	Flyweight	インスタンスを共有しコストを節約する
7	Proxy	オブジェクトへのアクセスを制御するために処理を代理人に任せる

振る舞いに関するパターン(1/2)

Agility for Business
Technologic Arts

- クラス、オブジェクトの通信に関する(全11パターン)
  アルゴリズムとオブジェクト間での適切な責任分担の方法を提供する

No.	パターン名	目的
1	Chain of Responsibility	要求に応じる役割をチェーン状につなぐ
2	Command	命令をカプセル化して再利用する
3	Interpreter	文法規則を表現する
4	Iterator	構造に順にアクセスする方法を提供する
5	Mediator	オブジェクト同士の結合度を低める

振る舞いに関するパターン(2/2)

No.	パターン名	目的
6	Memento	インスタンスの状態を戻せるようにする
7	Observer	状態の変化が自動的に通知され、更新される
8	State	状態にあわせて動作を変える
9	Strategy	アルゴリズムをカプセル化して交換可能にする
10	Template Method	特定の処理をサブクラスで行う
11	Visitor	構造と処理を分離する

ここまでのまとめ

- パターン全般について
- デザインパターンの意義
- GoFのデザインパターンの概要

Agility for Business
Technologic Arts

第3章　デザインパターン詳細

本日解説するパターン

Agility for Business
Technologic Arts

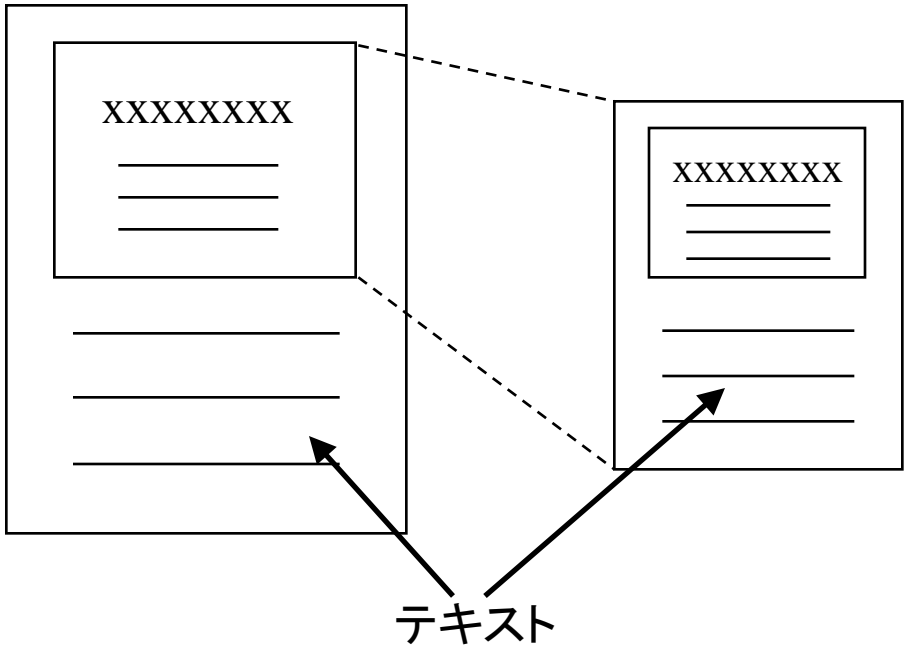
- Composite

Agility for Business
Technologic Arts

Compositeパターン

Compositeパターンの動機

Agility for Business
Technologic Arts



Composite（概要）

Agility for Business
Technologic Arts

特徴

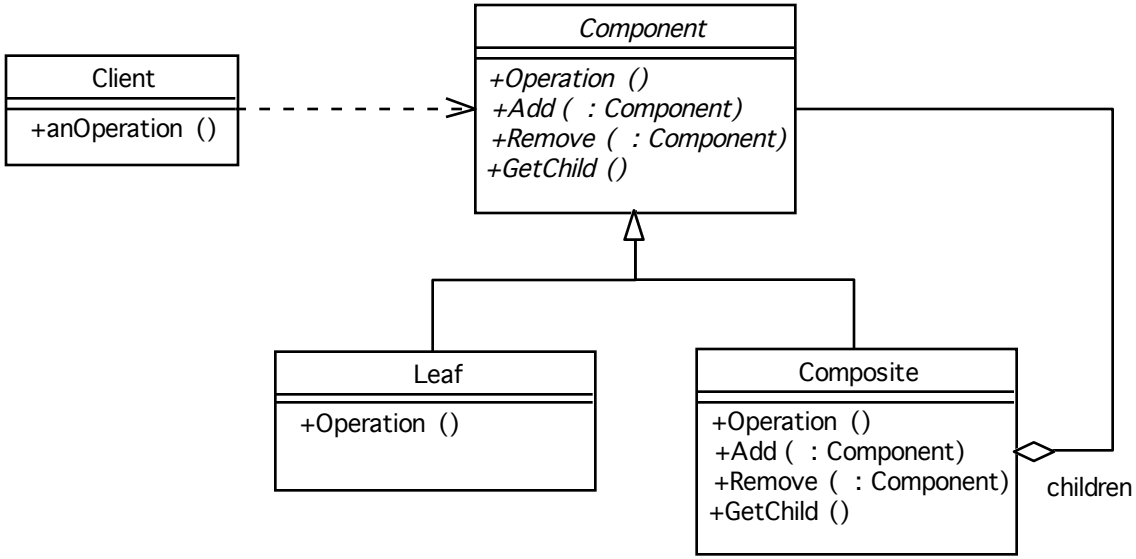
- 部分-全体階層を表現するために、オブジェクトを木構造に組み立てる
- 個々のオブジェクトとオブジェクトを合成したものを一様に扱う

結果

- クライアントは、個々のオブジェクトを同じように扱うことができる
- 複雑なオブジェクトを比較的簡単に構成しやすくなる

Composite（構造）

Agility for Business
Technologic Arts



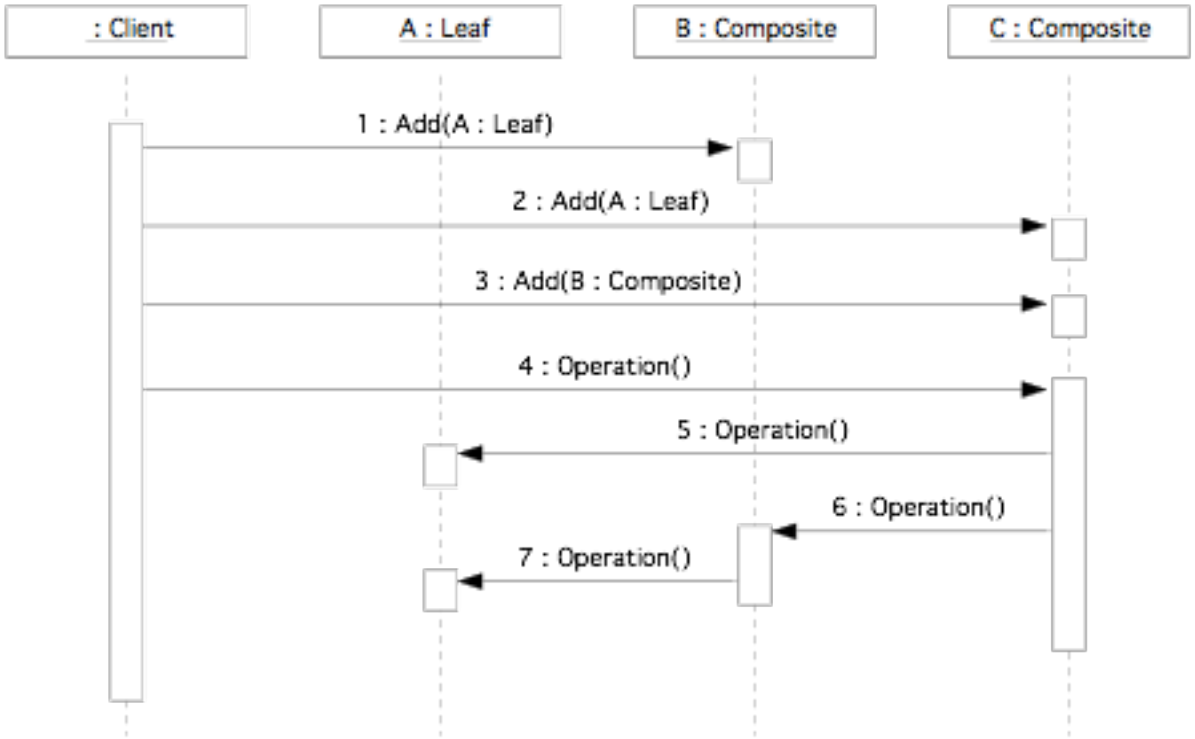
Composite（構成要素）

Agility for Business
Technologic Arts

- Component
  - 全てのクラスに共通のインタフェースを宣言する
  - 子にあたるComponentを管理するためのインターフェースを宣言する
- Leaf
  - 末端のオブジェクトを表す
- Composite
  - 子オブジェクトの入れ物にあたるオブジェクトを表す
  - 子オブジェクトに関する操作を実装する
- Client
  - Componentクラスのインタフェースを介してオブジェクトを操作する

Composite（振る舞い）

Agility for Business
Technologic Arts



Composite（サンプルモデル）

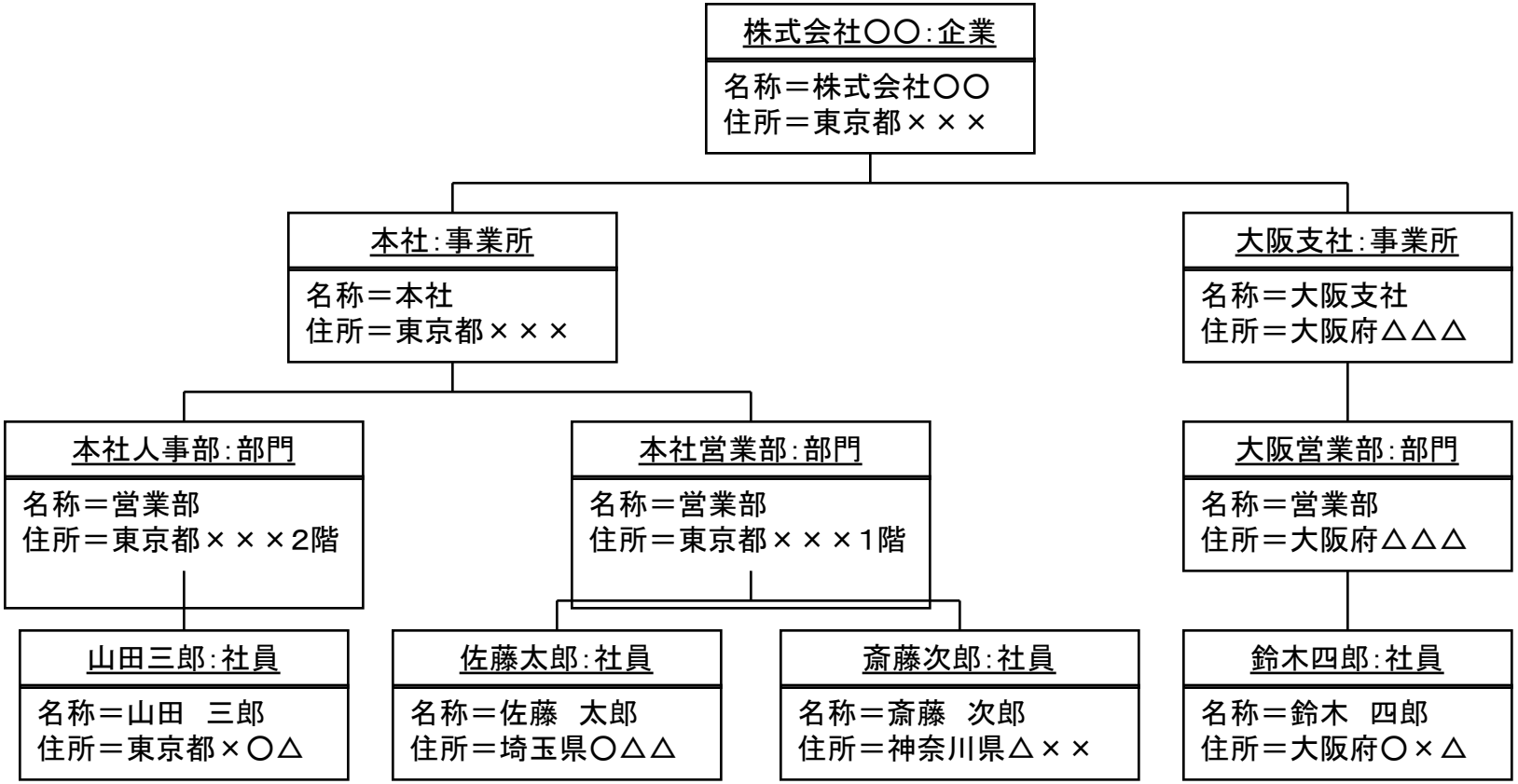
Agility for Business
Technologic Arts

- 組織管理を想定した分析を行う例

株式会社〇〇 （東京都×××）
　　本社 （東京都×××）
　　営業部 （東京都×××１階）
　　佐藤　太郎 （埼玉県〇△△）
　　斎藤　次郎 （神奈川県△××）
　　人事部 （東京都×××２階）
　　山田　三郎 （東京都×〇△）
　　大阪支社 （大阪府△△△）
　　営業部 （大阪府△△△ ）
　　鈴木　四郎 （大阪府〇×△）

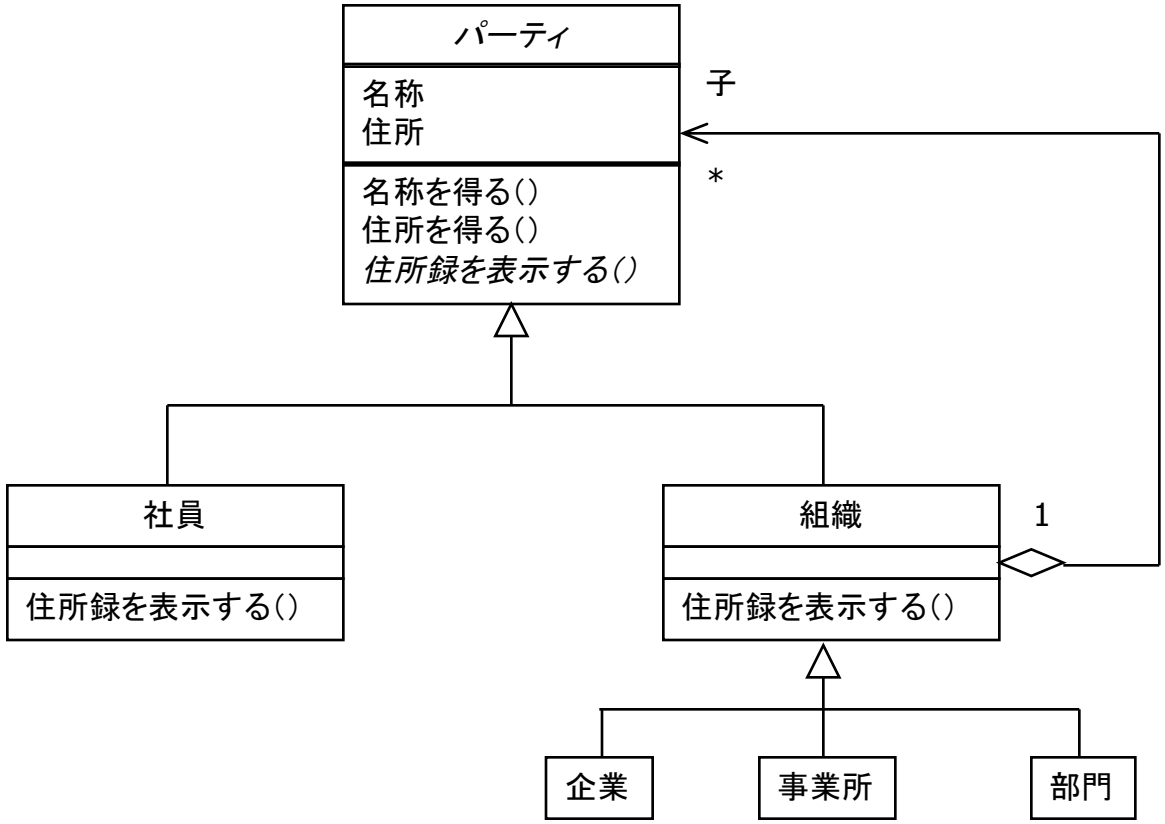
Composite（サンプルモデル）

Agility for Business
Technologic Arts



Composite（サンプルモデル）

Agility for Business
Technologic Arts



コード1

Agility for Business
Technologic Arts

```
import java.util.ArrayList;
import java.util.List;

public class Study50 {
public static void main(String[] args){

        Organization organization = new Organization("営業部","東京都港区南青山
");
        Staff staff = new Staff("田中一郎","東京都文京区");
        organization.add(staff);

        staff.showAddress();
        organization.showAddress();

        Staff staff2 = new Staff("鈴木一郎","東京都板橋区");
        organization.add(staff2);
        organization.showAddress();

}
}
```

コード2


```
class Party {
        String name;
        String address;
        Party(String name, String address) {
                this.name = name;
                this.address = address;
        }
        void showAddress(){
                System.out.println(address);
        }
}
```

コード3

Agility for Business
Technologic Arts

```
class Staff extends Party {
        Staff(String name, String
address) {
                super(name,address);
        }
}
```

コード４

Agility for Business
Technologic Arts

```
class Organization extends Party {
        List<Party> parties;
        Organization(String name, String address) {
                super(name,address);
                parties = new ArrayList<Party>();
        }
        void add(Party p){
                parties.add(p);
        }
        void showAddress(){
                super.showAddress();
                for (Party p : parties){
                        p.showAddress();
                }
        }
}
```

Composite（演習問題）

Agility for Business
Technologic Arts

- 国、県、市の単位で病院の名前と住所の一覧を取得したいとします。
- 上位の組織を選択して病院の名前と住所の一覧を取得すると、その下に属するすべての組織にある病院の名前と住所の一覧が取得できます。

- この要件を満たすように、 Compositeパターンを使って分析を行って下さい。
- 分析した設計を基に、プログラムを作成して実行してください。

Agility for Business
Technologic Arts

第5章　まとめ

デザインパターンを正しく使うために

Agility for Business
Technologic Arts

- デザインパターンを無理に適用しないこと
  → デザインパターンを使う理由を検討
  → デザインパターンを使って利点があるかを検討
- デザインパターンを少しずつ適用すること
  → 最初から全てを使いこなそうとしないこと
  例えばTemplateMethodから理解し、利用してみる
- 周りの人と共通な理解を得るために議論すること
  → 一人だけでなく、周りも同じように理解していなければ
  なりません。共通の理解を得てこそ、プロジェクトの
  中で正しく使うことができます

Agility for Business
Technologic Arts

第6章　参考資料

書籍から情報を収集する(1/2)

Agility for Business
Technologic Arts

- Erich Gamma ほか(著), 本位田真一, 吉田和樹(監訳):『デザインパターン　改訂版』, ソフトバンク, 1999.
- テクノロジックアート(著)、長瀬嘉秀、今野睦(監修):『パターンモデリングガイド』, ピアソン, 2002 .
- John Vlissides(著), 長瀬嘉秀, 永田渉(訳):『パターンハッチング 実践デザインパターン』, ピアソン, 1999.
- Mark Grand(著), 原潔, 宮本道夫, 瀬尾明志(訳) :『UMLを使った Java デザインパターン』, カットシステム, 2000.
- 鈴木純一, 田中祐, 長瀬嘉秀, 松田亮一:『ソフトウェアパターン再考』, 日科技連, 2000.
- 結城浩:『Java言語で学ぶデザインパターン入門』, ソフトバンク, 2001.

書籍から情報を収集する(2/2)

Agility for Business
Technologic Arts

- PLoPD Editors著, 細谷竜一, 中山裕子(監訳):『プログラムデザインのためのパターン言語』, ソフトバンク, 2001.
- 結城浩:『Java言語で学ぶデザインパターン入門 マルチスレッド編』, ソフトバンク, 2002.

Agility for Business
Technologic Arts

第4回Java超入門
~デザインパターン基礎~
WEBプログラマーへの第一歩

テクノロジックアート
長瀬　嘉秀