マルチパラダイムデザイン

再利用性の高いアプリケーションの設計

高橋 晶(Akira Takahashi)
faithandbrave@longgate.co.jp
2014/05/24(土) Boost.勉強会 #15 札幌

自己紹介

- **高橋 晶(Akira Takahashi)**
- はてなID：faith_and_brave
- TwitterID：@cpp_akira
- Boost.勉強会 東京の主催者
- boostjp／cpprefjpサイトのコアメンバ
- 著書：C++テンプレートテクニック、C++ポケットリファレンス、プログラミングの魔導書 Vol.1と3
- **好きなことは、「抽象」「直交」「再利用」「分割統治」**

本日のお題

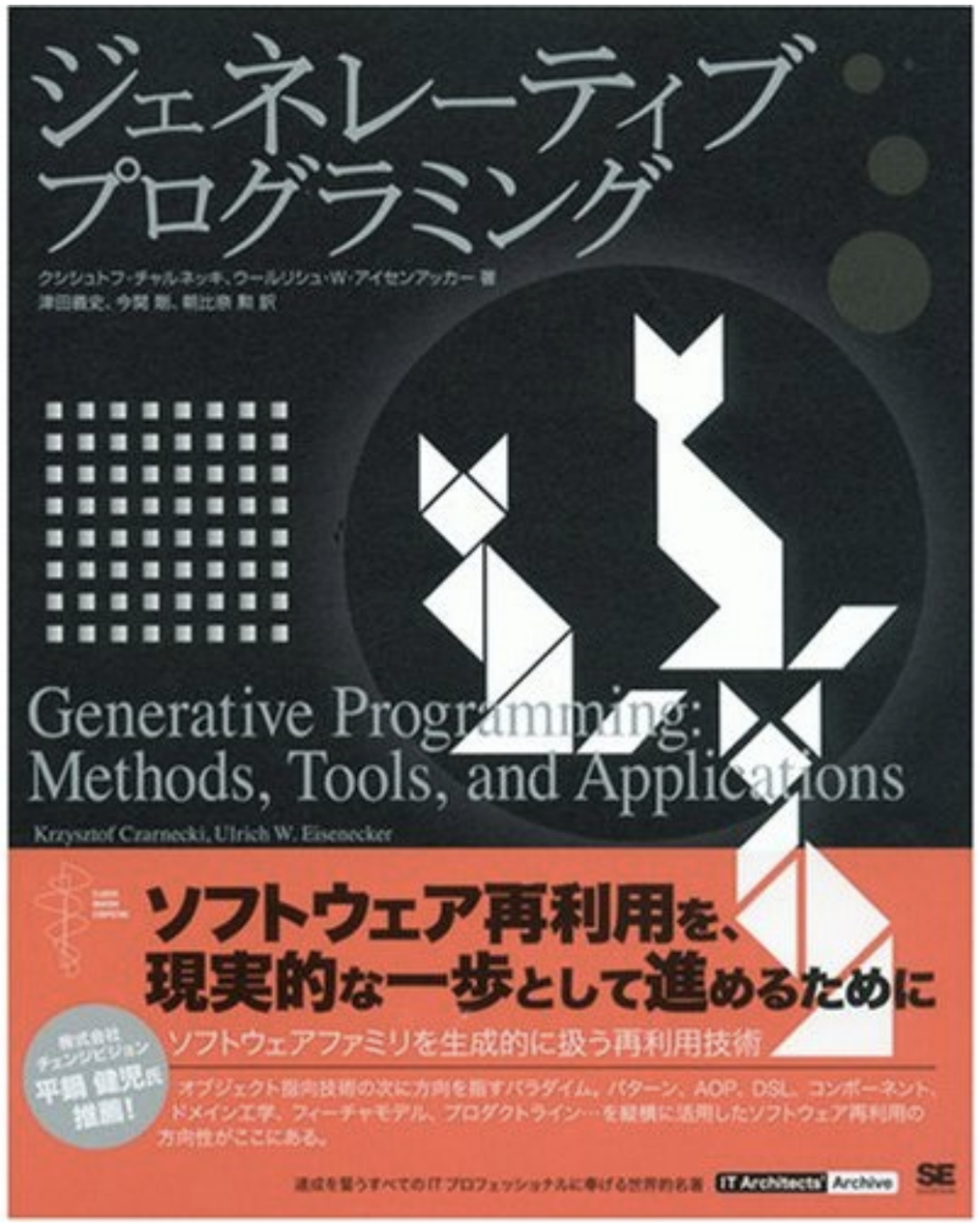
- **アプリケーションの設計**について話します。
  - ライブラリではない、ソフトウェアやゲームなどです。
  - 皆さんは普段、どのようなプログラム設計をアプリケーションに適用しているでしょうか。
  - 今日は、私が普段のアプリケーション設計に採用している、**マルチパラダイムデザイン**というものを紹介します。

再利用

- アプリケーションコードは、全てのコードがそのアプリケーション専用というわけではない。
- ほとんどは、汎用的なコードか、そのアプリケーションが属する(複数の)分野で共通利用できるコード。
- アプリケーションコードとしてべったり張り付いたコードを他のプロジェクトに再利用するのはめんどう。
- 再利用可能なコードとアプリケーション固有のコードを、きっちり住み分けするのが今日のお話。

文献


Multi-Paradigm Design for C++
James O. Coplien
新装版
マルチパラダイム
デザイン
ジェームズ・O・コプリン=著
平鍋 健児+金澤 典子+羽生田 栄一=訳
PEARSON
ピアソン・エデュケーション


ジェネレーティブ
プログラミング
クシシュトフ・チャルネッキ、ウールリシュ・W・アイセンアッカー 著
Generative Programming:
Methods, Tools, and Applications
Krzysztof Czarnecki, Ulrich W. Eisenecker
ソフトウェア再利用を、
現実的な一歩として進めるために
ソフトウェアファミリを生成的に扱う再利用技術
株式会社チェンジビジョン
平鍋 健児氏
推薦!
IT Architects' Archive
SE

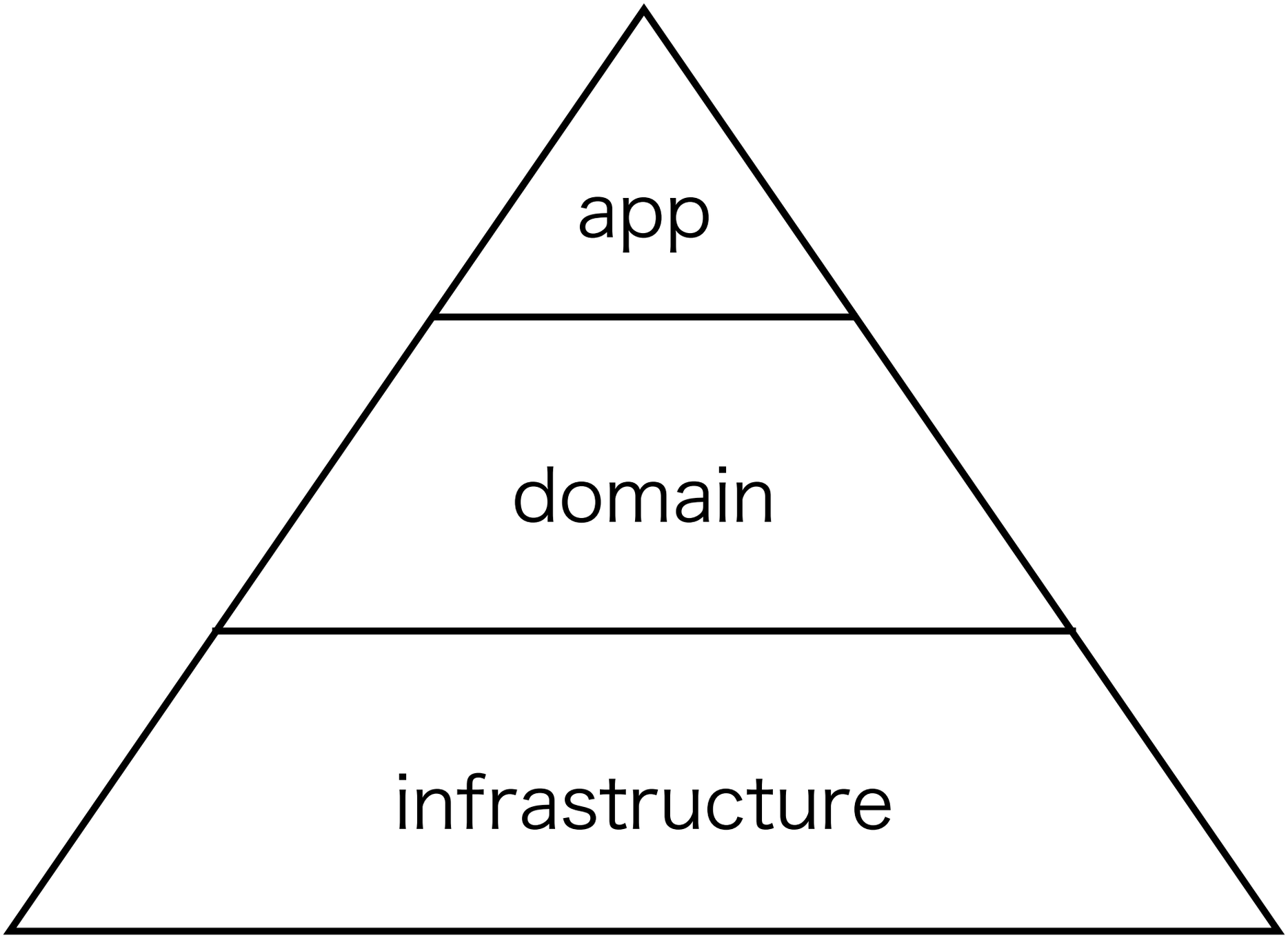
マルチパラダイムデザインの概要

- ドメイン駆動設計のアーキテクチャ(ディレクトリ構成)を採用する。
- アプリケーションを構成するプログラムを、ドメインという単位で分割し、部品の再利用性を高める。
- 各ドメイン／各機能を、共通性と可変性という考えの元に設計する。
- 部品の再利用性が高まることによって、類似アプリケーションを、実績あるコードを元に作り上げることができる。

ドメイン駆動設計

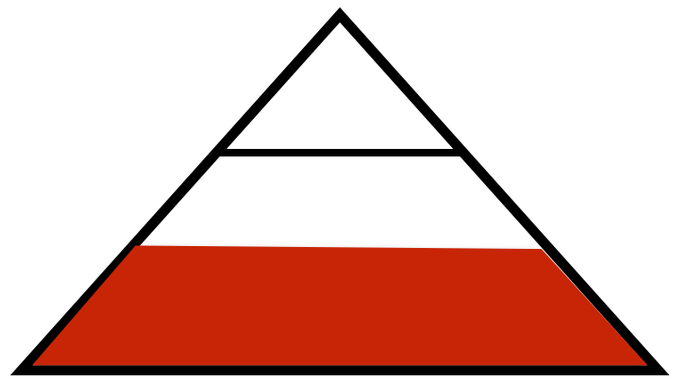
- Eric Evansが提唱した、アプリケーションをドメインに分割していく設計手法。
- プロジェクト内で共通の用語を使うユビキタス言語や、構成要素(エンティティ、値オブジェクト、リポジトリ、サービス等)が細かく規定されていたりする。
- 再利用は想定されておらず、一本のアプリケーションを効率的に作ることを目標にしている。
- 今回紹介する**私の**設計では、ドメイン駆動設計のレイヤー(ディレクトリ構成)のみを採用する。

ドメイン駆動設計のレイヤー構造



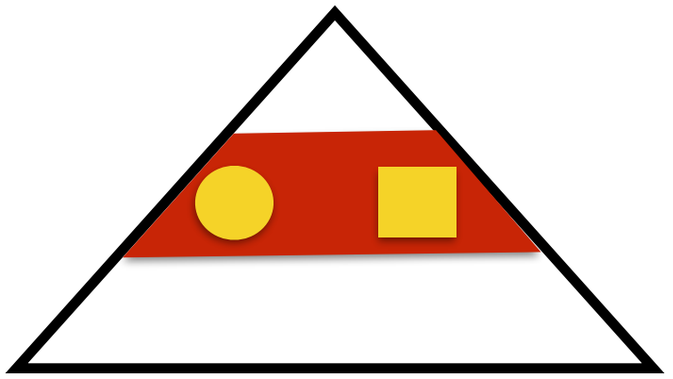
インフラストラクチャ

- アプリケーション全体で使用する、ベースとなるライブラリ、フレームワークを置くレイヤー。
- Boost、zlib、Ruby on Rails、.NET Framework等。



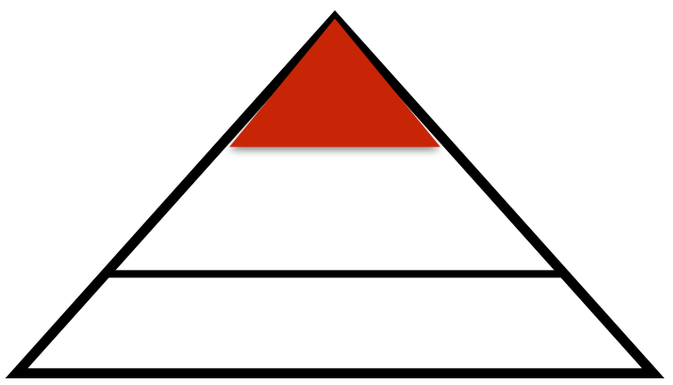
ドメイン

- アプリケーションを構成する部品を、できるだけ再利用可能な形で分割して置くレイヤー。ライブラリのようなもの。
- クラスは複数の機能をまとめたものだが、
  ドメイン(domain, 問題領域)はクラス・関数群をまとめた**機能ファミリー**。
- 「アプリケーションを作る！」という大きな問題を、
  ドメインという小さな問題群に分割する。



アプリケーション

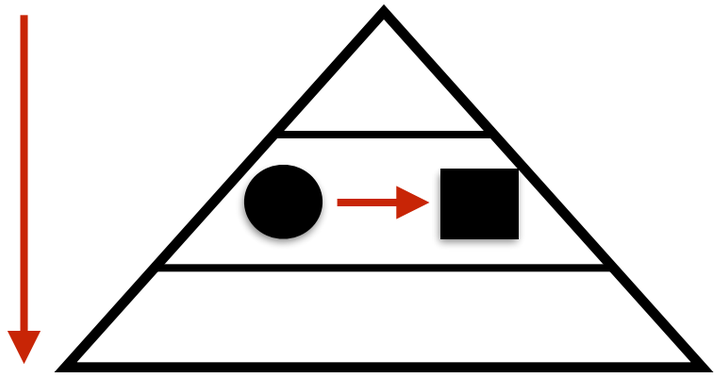
- 特定アプリケーションのみで必要な機能を置くレイヤー。
  - ユーザー情報、ビジネスロジック、戦闘ロジック、画面遷移など。
  - 本当はこの上に、見た目を制御する「プレゼンテーションレイヤー」があるが、それほど重要ではないので省略する。
- できるだけ多くの機能を再利用可能な形でドメインに分割するので、アプリケーションレイヤーは薄くなる。



依存関係

- レイヤーとしては、上のレイヤーから下のレイヤーへの依存のみ。
    - アプリケーションはドメイン、インフラストラクチャに依存するが、ドメインがアプリケーションに依存してはならない。

- ドメイン間は依存する場合がある。
  相互依存にならないよう、図に書き起こして依存関係を整理すること。



抽象化のためのドメイン分割

- ドメイン分割をすると、そのドメインのユーザー(アプリケーションや他のドメイン)へのインタフェースが、自然と抽象的になる。
  - 実装詳細をユーザーに直接使わせない。
- 抽象化が当たり前のように行われるようになる。

ここまでで大事なこと

- アプリケーションを構成するプログラムの
  レイヤーを意識する。
- 大きな問題(アプリケーションを作る！)を
  小さな問題群(ドメイン)に**分割統治する**。

再利用性を意識する

- ドメイン分割を行い、抽象的になったドメイン機能は、
  どのようにその機能が使われるか、さらに想定を広げる必要が出てくる。
- マルチパラダイムデザインでは、機能を共通性と可変性という特性で分析して設計する。
- **共通性(commonality)**とは、あらゆる目的に同じ使い方ができる性質。
- **可変性(variability)**とは、状況によって使い方が変わる性質。

共通性のためにできること

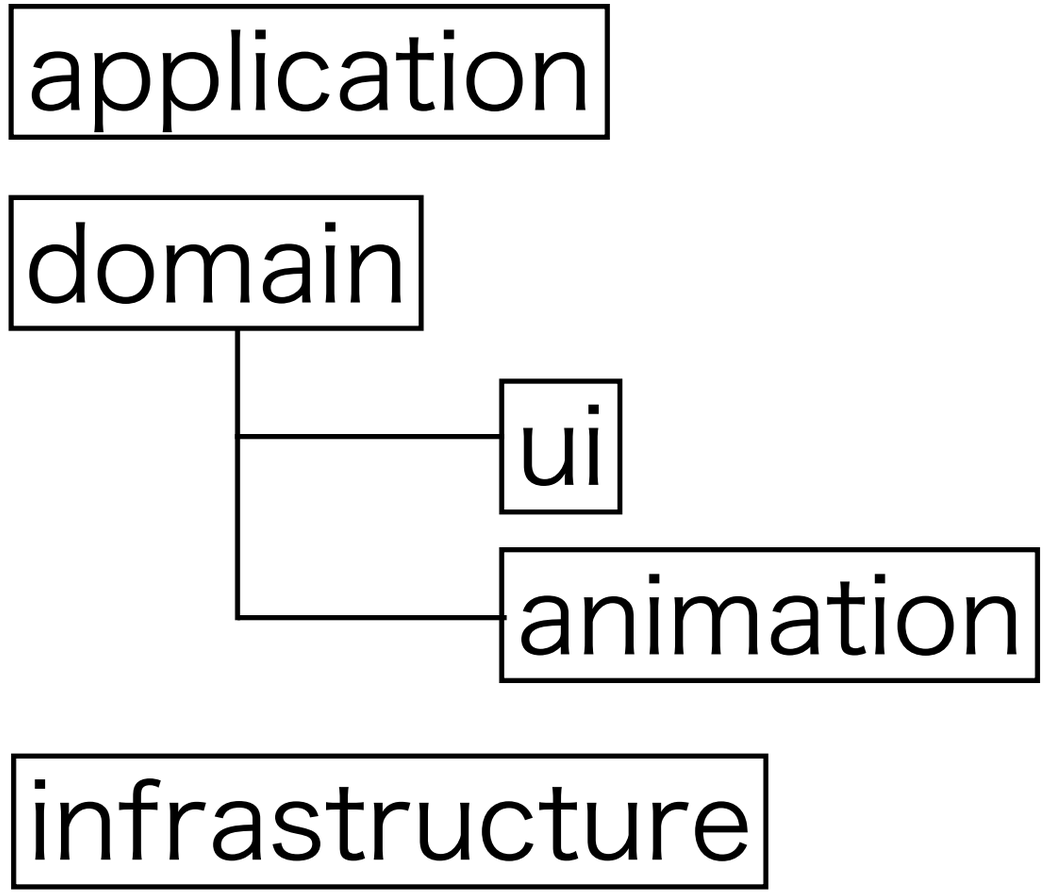
- 共通性を表現するためには、C++では継承やテンプレート、関数オーバーロードといったものを使用できる。
  - たとえば、`std::vector`は、あらゆる要素型に同じ操作方法を提供するという共通性を表現している。
- その他、プリプロセッサによるバックエンドの自動切り替えも、共通性のために使用できる。
  - たとえば、`boost::thread`は環境によって、pthreadによる実装とWindows APIによる実装を、自動的に切り替えている。ユーザーに提供するインタフェースを共通化している。

可変性のためにできること

- 可変性を表現するためには、C++では実行時多態性や、テンプレートのポリシークラス、トレイト、関数オブジェクトを使用できる。
- 与えられるパラメータによって動作を変えることは、たとえばstd::sort()が行っている。与えられる関数オブジェクトによって、比較の動作が変わる。
- 可変性を意識して設計したドメインは、他のプロジェクトへの再利用がしやすくなる。

ケーススタディ

- 以下の構成になっているドメインの設計を考える。



ケーススタディ

- ウィンドウに何らかのアニメーションをさせる

domain/ui/Window.h

```
namespace ui {
  class Window;
}
```

domain/animation/MyAnimation.h

```
namespace animation {
  class MyAnimation {
  public:
    void start(ui::Window& window);
  };
}
```

- いま、animationドメインはuiドメインに依存している

ケーススタディ

- ドメイン間の依存関係を断ち切る可変性を導入する

domain/animation/MyAnimation.h

```
namespace animation {
  class MyAnimation {
  public:
    template <class Window>
    void start(Window& window);
  };
}
```

- これで、animationドメインのみを他プロジェクトに再利用できる。

ケーススタディ

- 終了イベントを受け取る可変性を導入する。

```
namespace animation {
  class MyAnimation {
    std::function<void()> callback_;
  public:
    template <class Window>
    void start(Window& window);

    template <class Window, class F>
    void start(Window& window, F callback);
  };
}
```

- これで、MyAnimationクラスが、複数の用途・状況に対応できるようになった。

どこまで想定するか

- 抽象化や再利用とは、どこまで想定して作ればいいだろうか。
- 目の前のアプリケーションの要件だけを見てドメイン分割、抽象化、可変性の導入をすると、設計が破綻しやすい。
- 再利用性を高めるには、プロジェクトの今後の方向性や、複数のマーケットまでを見通す目が必要になる。

何がマルチパラダイムなのか

- 従来のオブジェクト指向設計では、可変性を表現する方法が多態性に限定された。
- マルチパラダイムデザインでは、他のパラダイムを共通性と可変性に使用し、設計にさらなる柔軟性を持たせる。
- たとえば、テンプレートの「特定インタフェースを持っていれば継承関係がなくていい」という性質は、ドメイン間のやりとりをよりシンプルにする。

DSL

- DSL(domain specific language, ドメイン特化言語)は、特定の問題を効率的に解決するための小さな言語。
  - 昔はアプリケーション指向言語(AOL : application oriented language)と呼ばれていた。
- 広義には、プレーンテキストの言語のみならず、ツールも含む。SQLやUIエディタ、GraphvizもDSL。
- 問題領域にくわしい専門家(ゲームならプランナー)に、使ってもらったりする。

まとめ

- マルチパラダイムデザインでは、アプリケーションを構成するプログラムを、小さな問題(ドメイン)に分割する。
- 分割したドメインを、共通性と可変性に注目して設計し、再利用性を高める。
- 再利用性を高めることによって、アプリケーションコードを他のプロジェクトに使いまわせるようになり、他のプロジェクトを、高い品質のコードを元にして作れるようになる。

みんな読もう！

- **あのCryoliteさんも2004年時点で読んでるぞ！**

Cry's Diary

プロフィール

Cryolite

2次元中毒患者（末期）．C++偏執狂．

About this blog

カウンター

カレンダー

<< 2004/04 >>

日	月	火	水	木	金	土
				1	2	3
4	5	6	7	8	9	10

<前の日 | 次の日>

2004-04-04 本2冊

[Software Engineering][書籍]マルチパラダイムデザイン

"マルチパラダイムデザイン"(asin:4894712989)を購入．さわりだけしか読んでないですがあれですね．かなり抽象的な話に始終していて難しいですね．まぁ，自分がいかにSoftware Engineeringを知らないか思い知らされますね．しかし，この本「この本ではC++で扱えるパラダイムをパラダイムと呼ぶ」とか書いてますけど，かなり大きく出ましたねw．個人的にはAspect-Orientedにも若干興味があるので，そこら辺との絡みというか「AOPだとこういう風に書けるけれど，C++はAOPを言語レベルでサポートしてなくて（´・ω・)ショボーン」な感じの内容があればうれしかったんですが．後，generic programmingというかtemplateとの絡みを期待していたのですが，流し読みした限りではあまり多くはなさそう．個人的に，C++にはgenericsの部分の効力をかなり評価していますので．

- **※ただしピアソンのため、絶版です。中古で…**