「大事は小事の組み合わせ出来ている。ここまでは同意だ。故に小事を疎かにしてはならない。これは反対だ。小事まで目を配ろうとするから破滅する。大事を為すためには、小事を任せられる人材を持つこと。これが大事を為す者の覇道だ」

指示者と作業者を分割統治して
多重継承のような柔軟な組み合わせを！

継承の多用でカオスになってアレしてコピペしないためのパターンです。考え方としてはテンプレートメソッドの抽象メソッド部分を差し替え可能な振る舞いオブジェクトに移譲するということです。オブジェクト指向に慣れている人であれば知らず知らずのうちに利用しているのではないかと思います。実態は「ポリモーフィズムの例題その1」という感じの基本的なパターンですしね。とりあえず、同じ条件のif文が至るところに現れたら、こいつを適用出来ないか？を考えると良いでしょう。こいつを取り扱える構造になったら、次の構造への発展も望めます。

グループカラー：ブルー

憂鬱である。戦争は終わった。しかし、戦災という名のもう一つの戦争が終わる気配は無い。自分は何のために戦ってきたのか。何のために、同胞を死地へ送ってきたのか。いくら自問自答を繰り返したところで答えは出そうにない。明日など一向に見えない。昨日もそう思っていたのに今日はやってきた。明日もきっと、そうしてやってくるのであろう。毎日、問題は山積みになっていく。いっそ時が止まってくれれば良いのに……机に突っ伏していたブリッジたんは長い溜息と共に顔を上げる。そんな甘美な夢想をしたところで、問題は魔法のように消えてくれはしない。

「戦争は終わった？」

終わったのかも知れない。しかし、同胞達は戦争を終わらせるために死んでいったのではない。皆が笑って暮らせる平和な祖国の実現のために死んでいったのだ。我々の戦いは「戦争」などという政治家が決めた区切りで終わる訳ではない。問題は確実に一つずつ解決していくしかない。

当面の問題は国境の街で起きている敵国の残存兵による略奪行為である。近場の砦で待機している兵隊を送り込んで殲滅するのは簡単なのだが、折角勝ち取った休戦条約にヒビが入る恐れがある。無傷で捕えて敵国に送り返してやらねばならない。大きな溜息をつき、内線を回す。

「―――――――― ということで頼んだ」

彼女は部下の使い方が上手いことで有名である。多少の効率をロスしてでも、全ての手続きを一貫した手順に纏めることで応用を効かせているのである。彼女は「異なる事柄を任せる人材を選出」し「小さな命令の手続を統一」する。この二つを徹底した。この結果、この小さな命令を様々に組み合わせた大きな命令書を状況に応じて作成し、それぞれ別の人材に任せることで無限の処理に対応出来るようにしたのである。戦略のアキレス腱は、伝達である。その手続きさえ一貫していればミスは少なくなる。あとは、何を頼むのかによって選出する相手さえ間違わなければ充分な結果が得られるのだ。

彼女はよくある戦記物の軍人のように華々しい奇策で大勝するようなことはない。これから戦う相手をよく調べ、充分に勝てる戦力を用意して、正確に戦い、正確に勝つ。そういうタイプの軍人である。

しかし一度だけ、圧倒的な不利を覆して歴史的な勝利を勝ちとった会戦があった。砂漠の野戦である。敵兵5万の正規軍に対して、こちらは負傷兵を回収に来た衛生兵と義勇軍混じりの護衛、合わせて500であった。加えて相手は戦車を数多く配備していたのに対して、こちらは輸送用のバギーだけ。彼女は

「本作戦の目的は負傷兵の回収であり相手との交戦ではない」とした上で、衛生兵の隊長に以下のように指示を出した。

「負傷兵を全て回収してまわれ。回収の指示についてはタイミングを見計らって無線で“実行しろ”と指示を出す。回収が完了したことを確認したら、次に“撤退しろ”と指示を出す。指示を理解した時、負傷兵が動けそうならば自分はB地点で降りてA地点まで離脱しろ。負傷兵が動けそうになければA地点までバギーで離脱しろ」

次に義勇兵の隊長には以下のように指示を出した。

「バギーに乗ったら敵戦車のラインに近付き、ラインの手前目掛けてパイナップルを投げろ。投げるタイミングについては無線で“実行しろ”と指示を出す。完了したことを確認したら、次に“撤退しろ”と指示を出す。指示を理解したらA地点を経由してC地点まで離脱しろ」

まず、敵のラインが遠いうちに義勇兵を全てバギーから降ろし、衛生兵全てにバギーが行き渡るようにした。護衛に割くリソースを省略し、回収にかかる時間を短縮した。次に全ての義勇兵をバギーに乗せてパイナップルを投げさせ、C地点まで撤退させた。パイナップルは砂を巻き上げ敵軍の視界を奪った。砂が晴れ、いざ追撃せんと敵軍の戦車はC地点まで直行しようとした。それはつまりB地点を経由することを意味していた。すると、次から次へと敵軍の戦車の足元が爆発し、その機能を停止していった。

B地点で降ろされた負傷兵には、以下の指示が出されていたのだ。

「バギーに積んである地雷を設置しろ。設置するタイミングについては無線で“実行しろ”と指示を出す。次に“撤退しろ”と指示を出す。指示を理解したらA地点で衛生兵と負傷兵を回収してC地点まで離脱しろ」

これにより敵軍は戦車の修理と回収のために足止めを余儀なくされ、彼らは予めブリッジたんが呼んでおいた援軍により一兵残らず駆逐されることとなった。この勝利により一躍時の人となったブリッジたんは参謀本部に呼ばれ、戦線を裏で支える任に着き、着実に勝利を重ね、軍神の名を欲しいままにしていく。

最近、彼女の酒量が増えた。それは戦災の辛さから逃げるための酒なのか、戦争が終わり世が着実に良くなっていく喜びを深く味わうための酒なのか、それとも死なせてしまった部下達への弔い酒なのか。酔うと彼女は無二の宝である部下達にこう言う。

「大事は小事の組み合わせで出来ている。ここまでは同意だ。故に小事を疎かにしてはならない。これは反対なのだ。小事まで目を配ろうとするから破滅するのだ。大事を為すためには、小事を任せられる人材を持つこと。これが大事を為す者の覇道なのだよ」

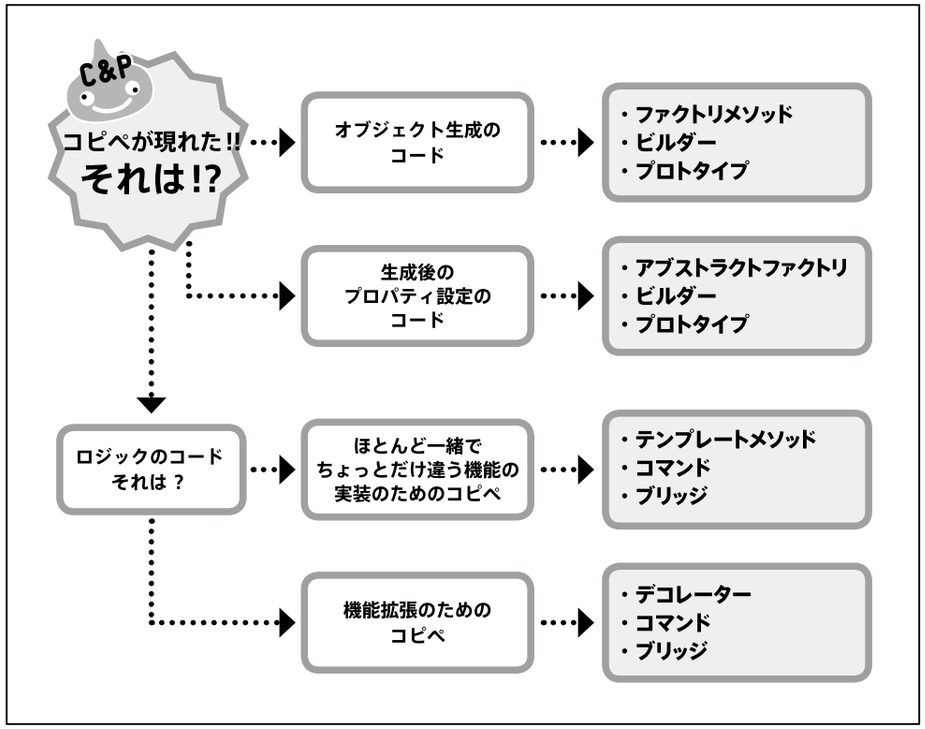
自分にそう言い聞かせているのだろうか。友が、家族が安心して生きられる明日のために、見えない明日を抉じ開けようとして、彼女は今日も指令を飛ばすのである。

```php
<?php
/**
 * Abstraction - Implementorを保持し
 * Implementorを使った基本的な機能が実装だ
 * RefinedAbstraction - Abstractionの機能追加
 * Implementor - Abstractionが使用する
 * インターフェースを規定するぞ
 * ConcreteImplementor - 具体的なImplementorを
 * 実装するぞ
 */
// Abstraction
class BridgeSample {
    private $impl = null;

    public function __construct ( $impl ) {
        if ( !( $impl instanceof Implement ) )
            throw new Exception ( 'こいつは無理な
相談なのだよ' );
        $this->impl = $impl;
    }

    //ImplementorのdoMethod()を使うのだ
    //Implementorを差し替えるだけで実装が
    // 切り替わるということなのだよ
    public function doMethod () {
        return $this->impl->doMethod ();
    }
}

// RefinedAbstraction その1なのだよ
class BridgeSub1 extends BridgeSample {
    public function doMethod () {
        return "Sub1|" . parent::doMethod ();
    }
}

// RefinedAbstraction その2なのだよ
class BridgeSub2 extends BridgeSample {
    public function doMethod () {
        return "Sub2|" . parent::doMethod ();
    }
}

// Implementorなのだよ
interface Implement {
    public function doMethod ();
}

// ConreateImplementor その1なのだよ
class ImplementSample1 implements Implement {
    public function doMethod () {
        return "実装1";
    }
}

// ConcreteImplementor その2なのだよ
class ImplementSample2 implements Implement{
    public function doMethod () {
        return "実装2";
    }
}

function say($s) { print "$s\n"; }

// 以下同じdoMethod()を使っているが
// RefinedAbstractionで機能が拡張され
// ConcreteImplementorを差し替えることによって
// 別の実装処理がされるのだよ
$parent = new BridgeSample ( new ImplementSamp
le2 () );
say ( $parent->doMethod() == '実装2' ? "OK" :
"NG" );

$subclass1 = new BridgeSub1 ( new ImplementSam
ple1 () );
say ( $subclass1->doMethod() == 'Sub1|実装1' ?
"OK" : "NG" );

$subclass1_2 = new BridgeSub1 ( new ImplementS
ample2 () );
say ( $subclass1_2->doMethod() == 'Sub1|実装2'
? "OK" : "NG" );

$subclass2 = new BridgeSub2 ( new ImplementSam
ple1 () );
say ( $subclass2->doMethod() == 'Sub2|実装1' ?
"OK" : "NG" );

$subclass2_2 = new BridgeSub2 ( new ImplementS
ample2 () );
say ( $subclass2_2->doMethod() == 'Sub2|実装2'
? "OK" : "NG" );
```


01 どんな時にどのパターンを適用するの？

まず、概念クラス図を描いてシステムを構造化する時にパターンが適用出来そうな箇所を探す…… というのが王道ではあるのですが、それは人によっては難易度が高い行為だったりします。そもそもデザインパターンの成り立ちについて想像すれば「オブジェクト指向プログラミングをしている時、ソースがカオスになり過ぎたので、こうすれば良かった、あーすれば良かった」というのが有るでしょう。それはソースがカオスにならなければ、別にデザインパターンなんて考える必要も無かったはずなので。つまり出来上がったソースの中から適用すると良くなりそうな場所を探すというのも有効な手段だったりします。それを積み重ねていくことで「こういう時にはこれが適用出来る」という経験が身についてくると思います。

以下に、今回登場したパターンの中から適用するべきパターンが見付かるかも知れないフローを用意しました。



上記の内容とはズレますが、本書では以下のようにグループカラーを決めています。
赤：生成に関するパターン 黄：振る舞いに関するパターン 青：構造に関するパターン
このグループカラーはでざぱたんの髪色にも反映されています。