

2018.03.10

PHPerKaigi 2018

SOLIDの原則って どんなふうにするの？

～ オープン・クローズドの原則編 ～

後藤 秀宣

hidenorigoto@gmail.com

公開・配布版



本題

SOLIDの原則ってどんなふうにするの？

～オープン・クローズドの原則編～

なぜSOLIDの原則？

なぜSOLIDの原則？

今現在のソフトウェア開発においても
なお、とてもよく効果を発揮する設計原則だから

なぜオープン・クローズドの原則？

なぜオープン・クローズドの原則？

**SOLIDの原則全体が目指している
「良い設計」を
もっともよく反映している原則だから**

このトークの目標

このトークの目標

オープン・クローズドの原則を理解する

このトークの目標

オープン・クローズドの原則を理解する

**オープン・クローズドの原則を意識して
コードを書ける**

ストーリー

ストーリー

わたしは新人PHPer。

PHPでのプログラミングは一通りできるようになり、
いよいよ現場で、小さなプロジェクトを担当することになった。

先輩からは、今の段階で学んでおくこととして、
コード設計についての知識や目を養ってほしいと言われている。

「コード設計」と言われても、はたして何をすることなのか。

今はぼんやりとしか分からない。

この機会に、きちんと学ぼうと思っている。

アプリの最初の要件

アプリの最初の要件

コマンドラインで使う
TODOアプリ

データベースに登録済みのTODO
を一覧表示する機能のみ

いわゆるMVCフレームワークは
使わない

アプリの最初の要件

コマンドラインで使う
TODOアプリ

TODOに
バリエーションがある

データベースに登録済みのTODO
を一覧表示する機能のみ

通常のTODO

GoogleCalendar
由来のTODO

いわゆるMVCフレームワークは
使わない

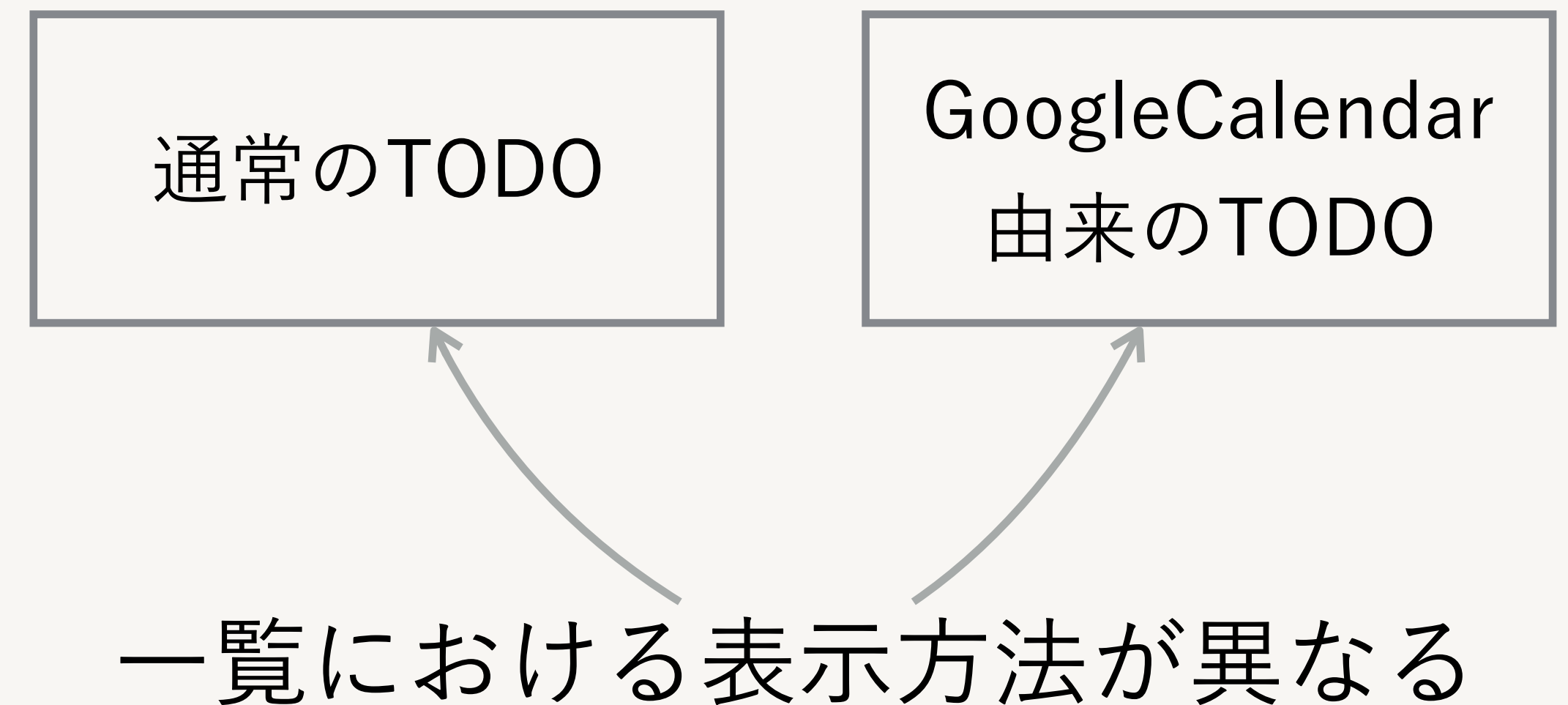
アプリの最初の要件

コマンドラインで使う
TODOアプリ

TODOに
バリエーションがある

データベースに登録済みのTODO
を一覧表示する機能のみ

いわゆるMVCフレームワークは
使わない



TODOの表示要件

TODOの表示要件

通常のTODO

[未着手] ○○について調査

<https://example.com/todo/1>

[完了] Bの実装

<https://example.com/todo/1>

[ステータス] タイトル

URL

TODOの表示要件

通常のTODO

[未着手] ○○について調査
<https://example.com/todo/1>
[完了] Bの実装
<https://example.com/todo/1>

[ステータス] タイトル
URL

Google Calendar由来のTODO

3/15 10時 A社訪問 (A社)
<https://example.com/todo/1>
[完了] C仕様打ち合わせ (会議室)
<https://example.com/todo/1>

日時 タイトル (場所)
URL

または

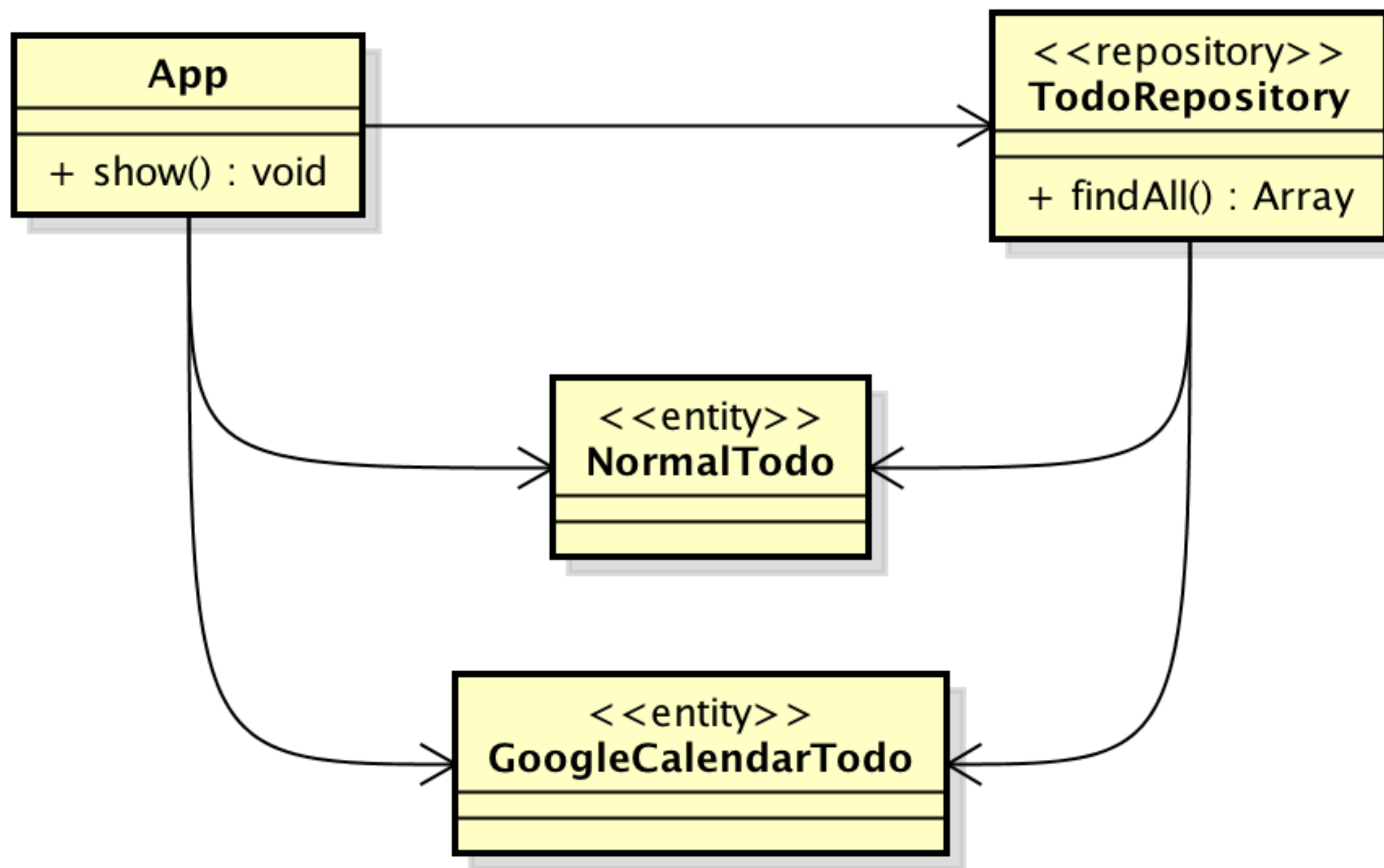
[完了] タイトル (場所)
URL

早速作ってみる

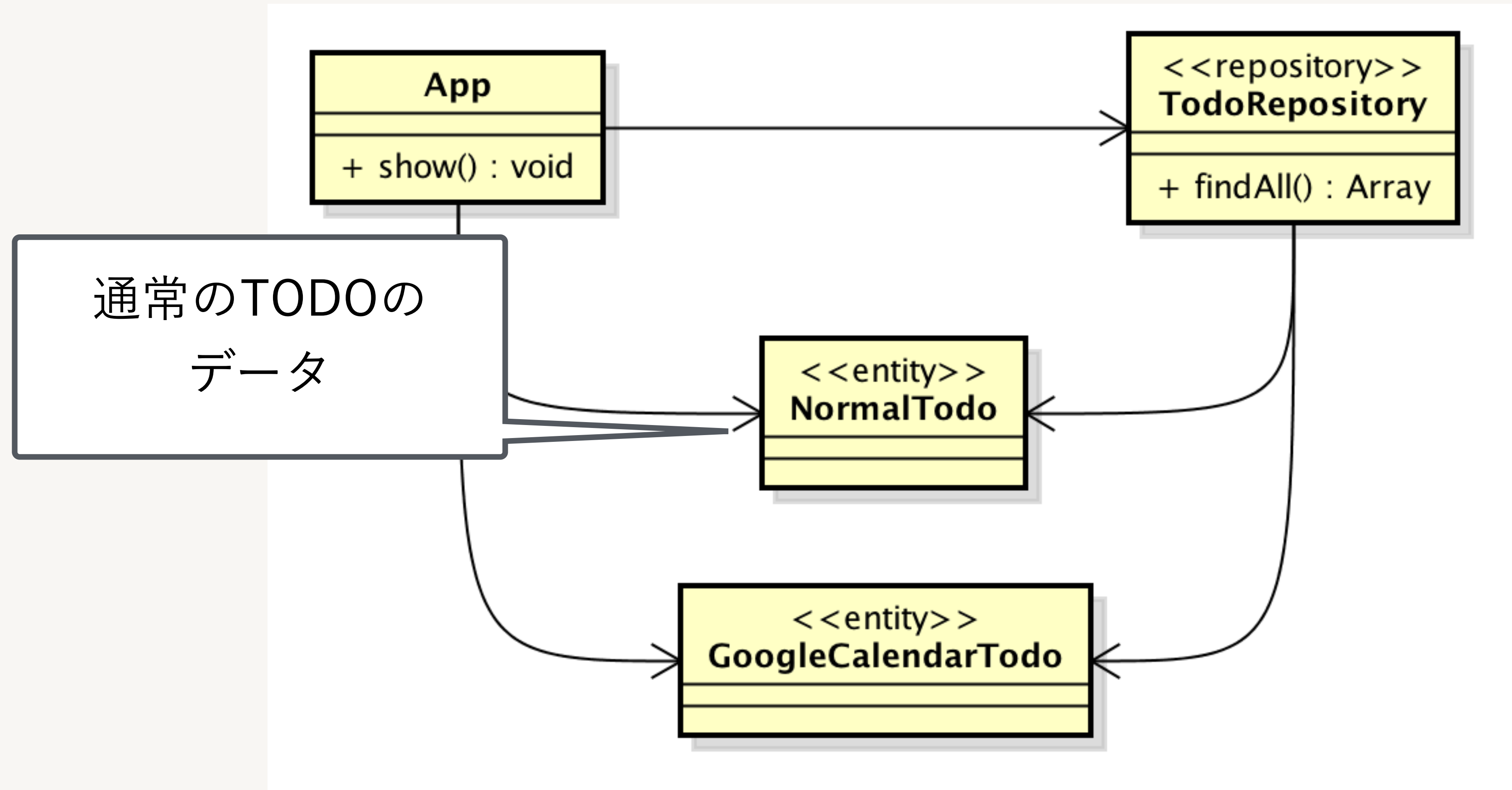
何となく作れそうッス
by 新人

クラスの設計

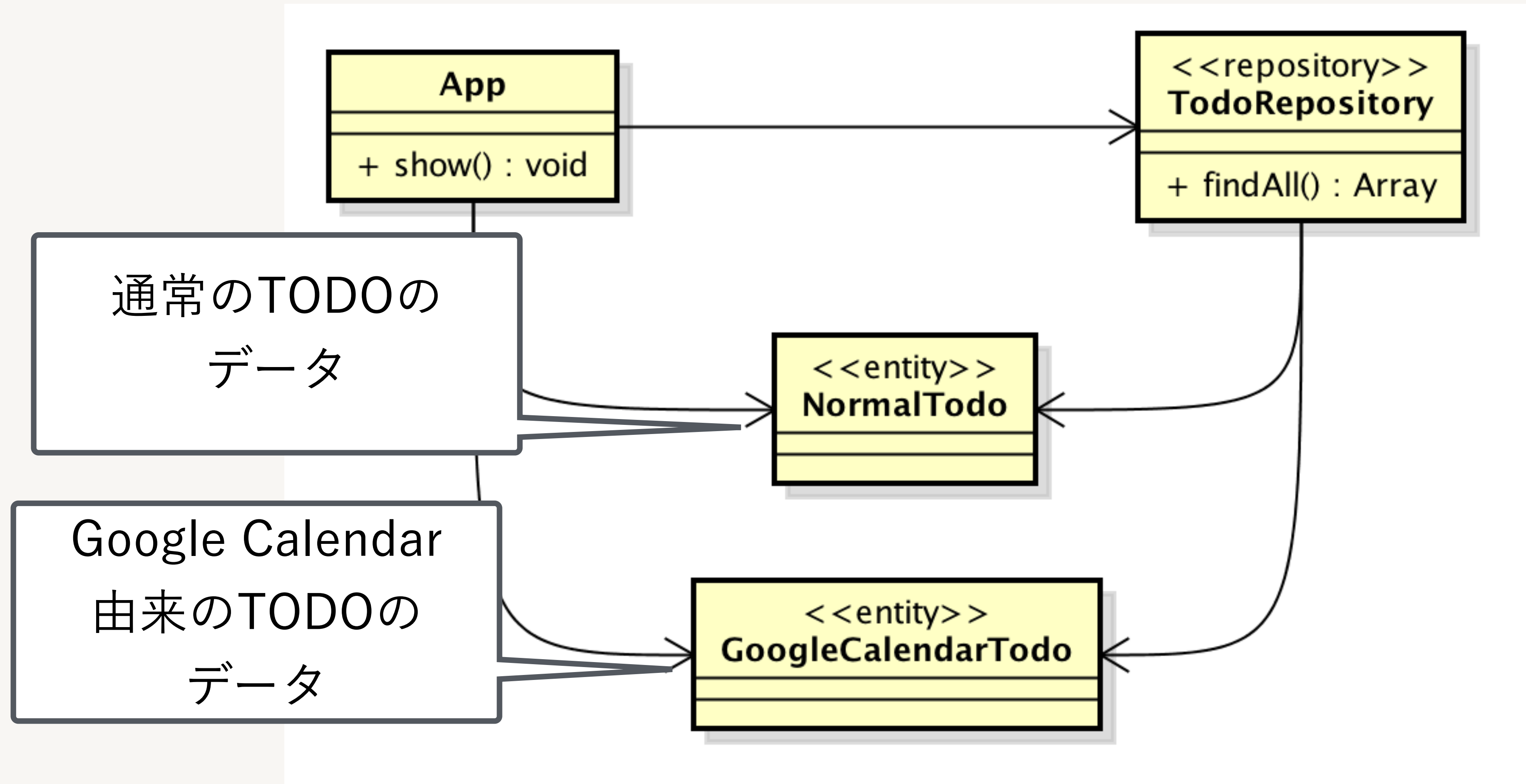
クラス的设计



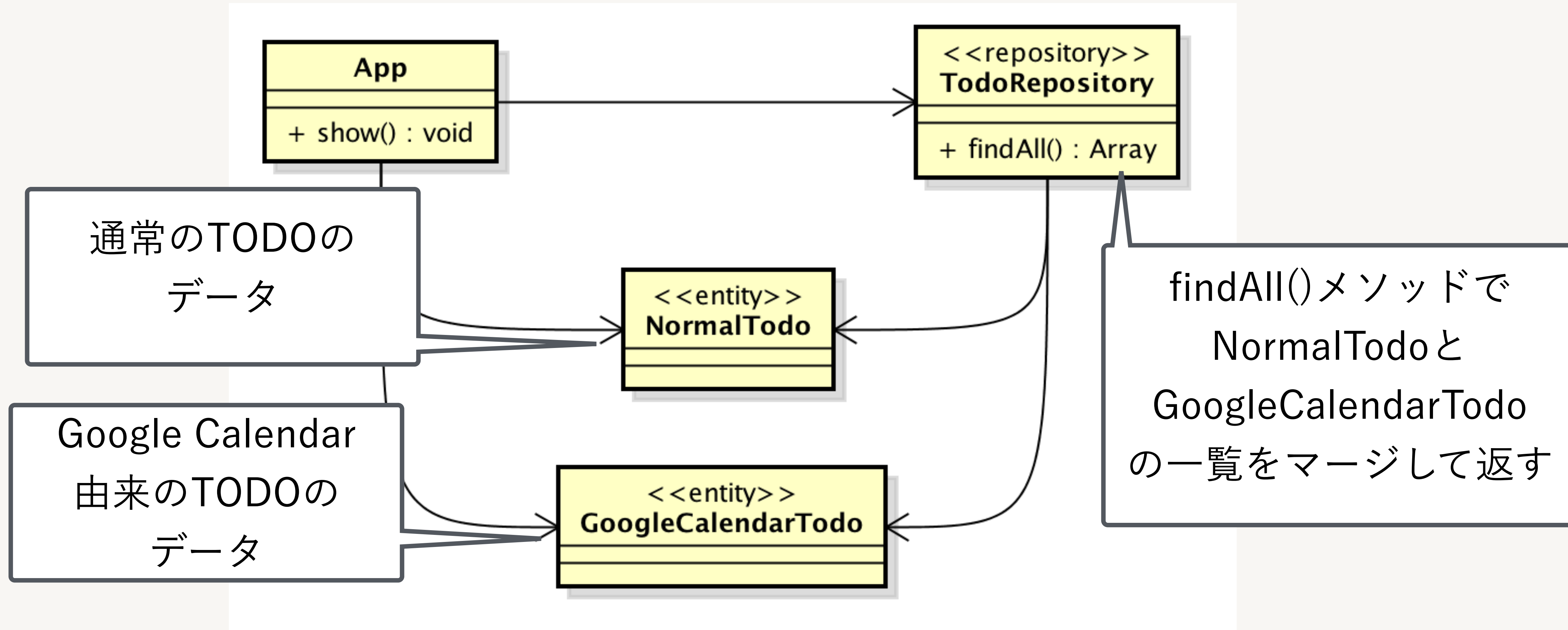
クラス的设计



クラス的设计

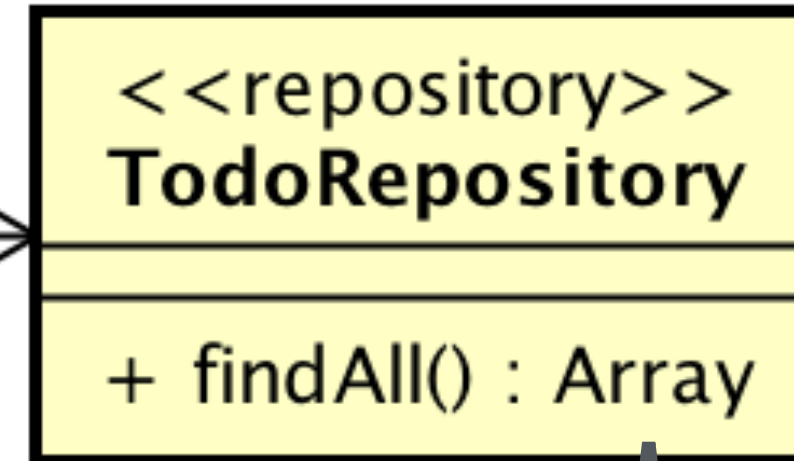
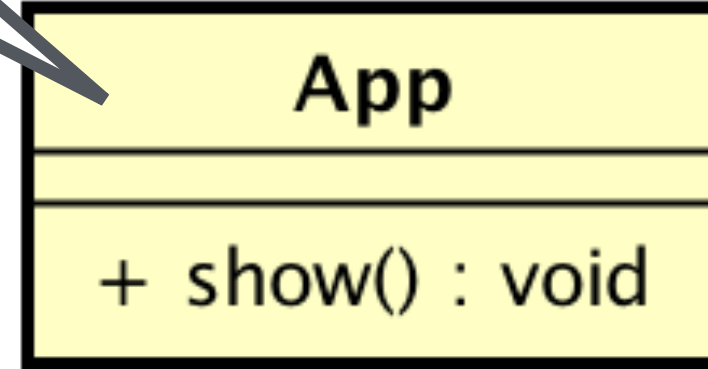


クラス的设计

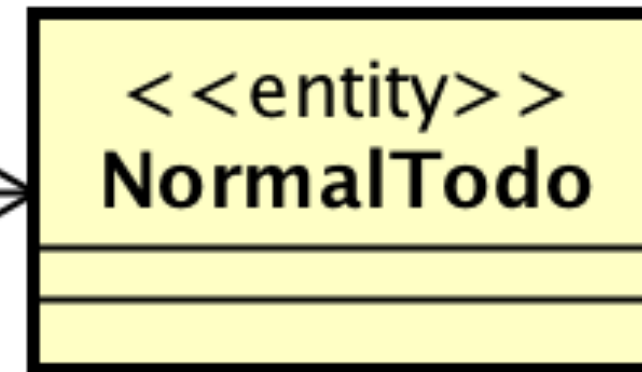


クラス的设计

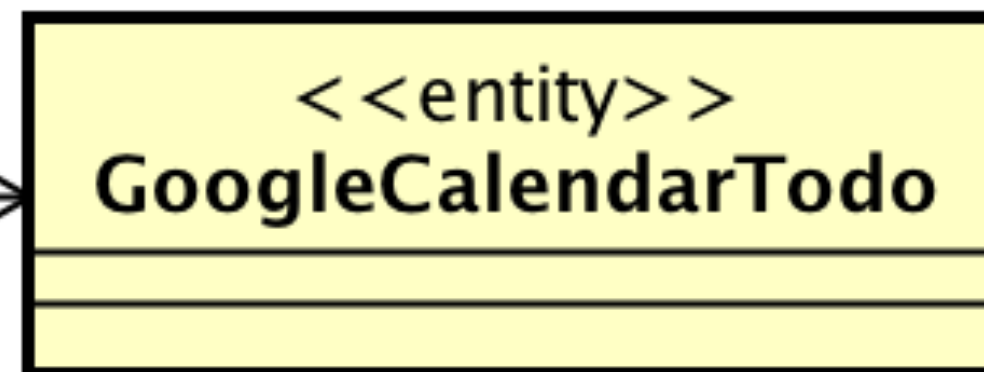
show()メソッドで
一覧表示



通常のTODOの
データ



Google Calendar
由来のTODOの
データ



findAll()メソッドで
NormalTodoと
GoogleCalendarTodo
の一覧をマージして返す

Appクラスのshow()メソッド

Appクラスのshow()メソッド

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        echo '-----';  
        echo $this->formatTitle($todo) . PHP_EOL;  
        echo $todo->getUrl() . PHP_EOL;  
    }  
}
```

Appクラスのshow()メソッド

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        echo '-----';  
        echo $this->formatTitle($todo) . PHP_EOL;  
        echo $todo->getUrl() . PHP_EOL;  
    }  
}
```

タイトル行表示

Appクラスのshow()メソッド

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        echo '-----';  
        echo $this->formatTitle($todo) . PHP_EOL;  
        echo $todo->getUrl() . PHP_EOL :  
    }  
}
```

URL行表示

AppクラスのformatTitle()メソッド

AppクラスのformatTitle()メソッド

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
    return $title;  
}
```

AppクラスのformatTitle()メソッド

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
  
    return $title;  
}
```

Todoごとに
ステータスの処理方法が
だいぶ異なるので、別メ
ソッドで。

AppクラスのformatTitle()メソッド

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
  
    return $title;  
}
```

Google Calendarの場合の
場所情報を追加。

AppクラスのformatStatus()メソッド

AppクラスのformatStatus()メソッド

```
private function formatStatus(string $status) :string {  
    $map = [  
        'pending' => '[未着手] ',  
        'running' => '[作業中] ',  
        'completed' => '[完了] ',  
    ];  
  
    return $map[$status];  
}
```

通常TODOのステータス値をラベルにマッピング

AppのformatStatusOfGoogleCalendar()メソッド

AppのformatStatusOfGoogleCalendar()メソッド

```
private function formatStatusOfGoogleCalendar(  
    GoogleCalendarTodo $todo) :string {  
    $now = new \DateTime();  
    if ($todo->getEndTime() > $now) {  
        return '[完了]';  
    } else {  
        return $todo->getEndTime()->format('n/j H時');  
    }  
  
    return '';  
}
```

完了日時のみで
判定

自分なりに設計して実装できた！

けど、コードが分かりづらくなっている気がする。

なぜこうなってしまうんだろう・・・

こんな時は、先輩にアドバイスをもらおう！

先輩からのコメント

先輩からのコメント

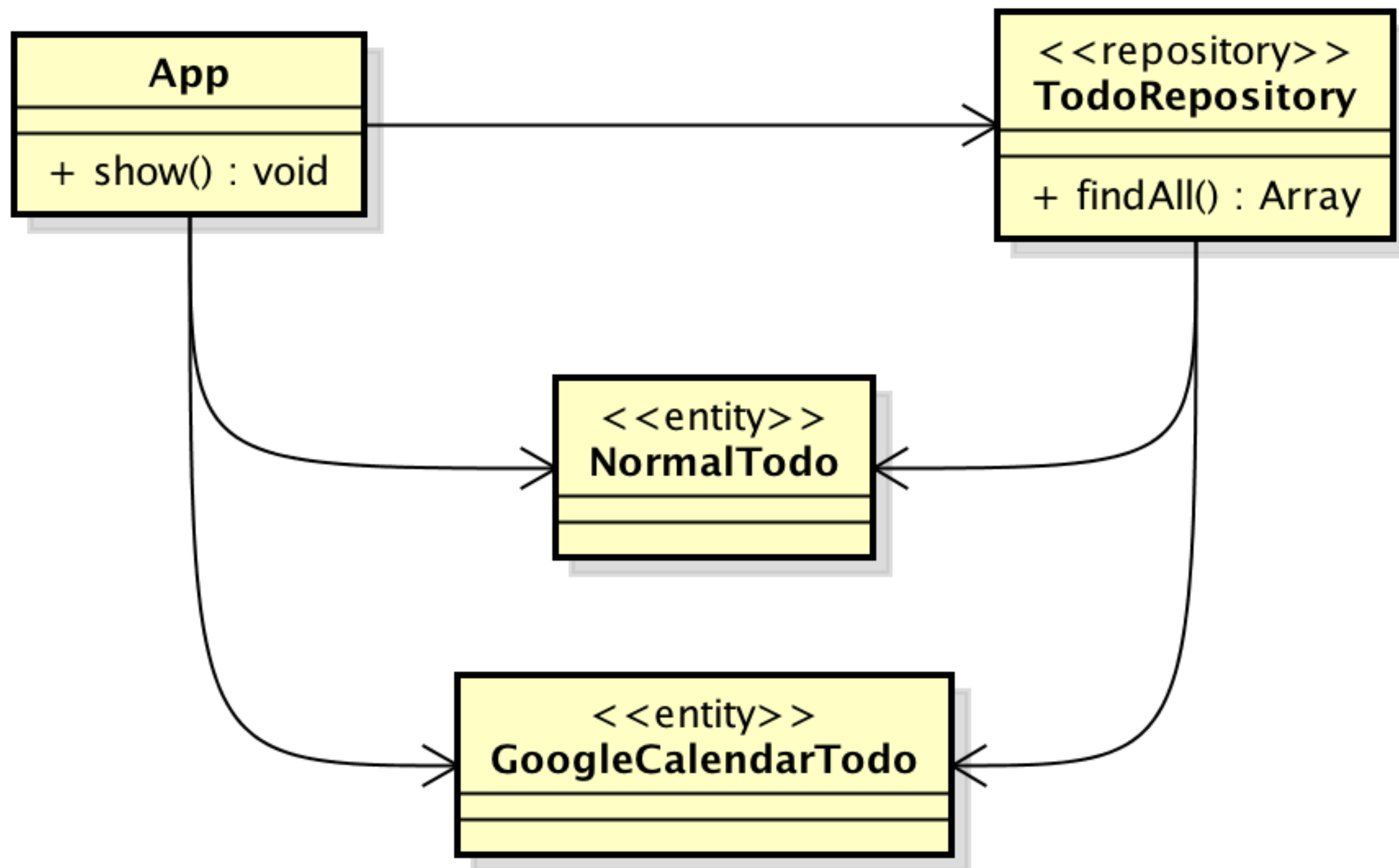
先輩です。

先輩からのコメント

先輩です。

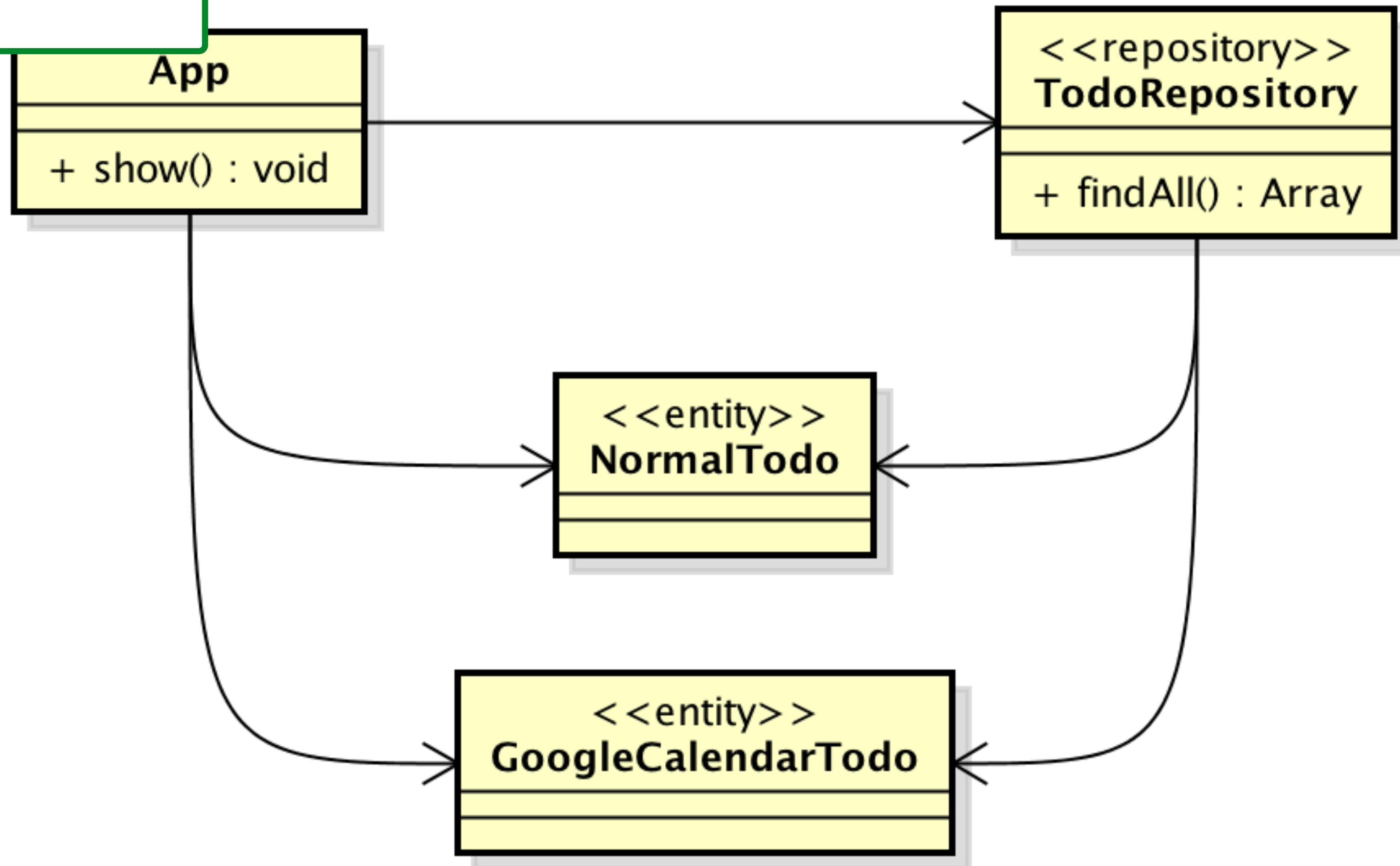
新人です！

クラス的设计

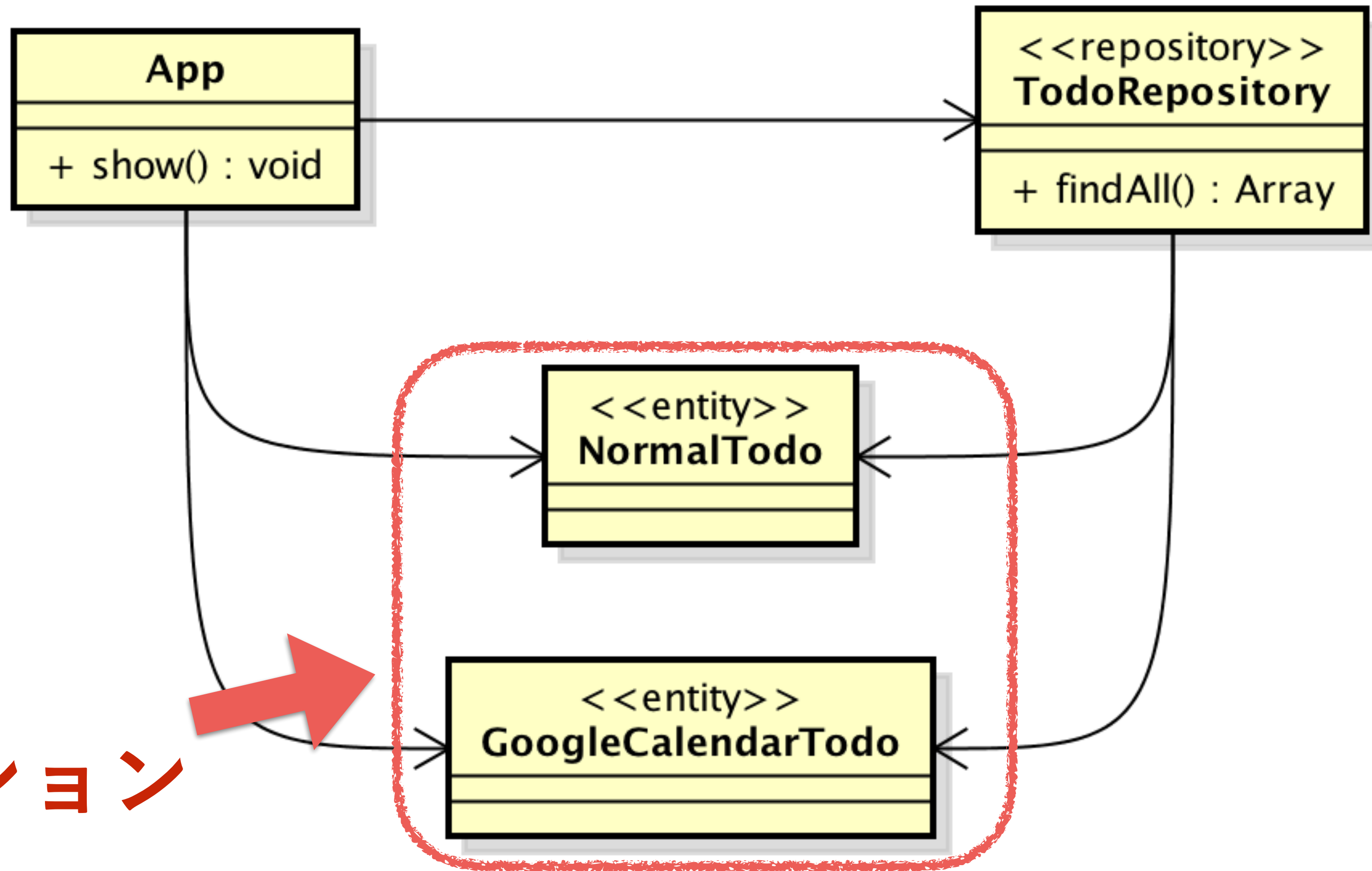


クラス的设计

全体の構造は
悪くないですよ

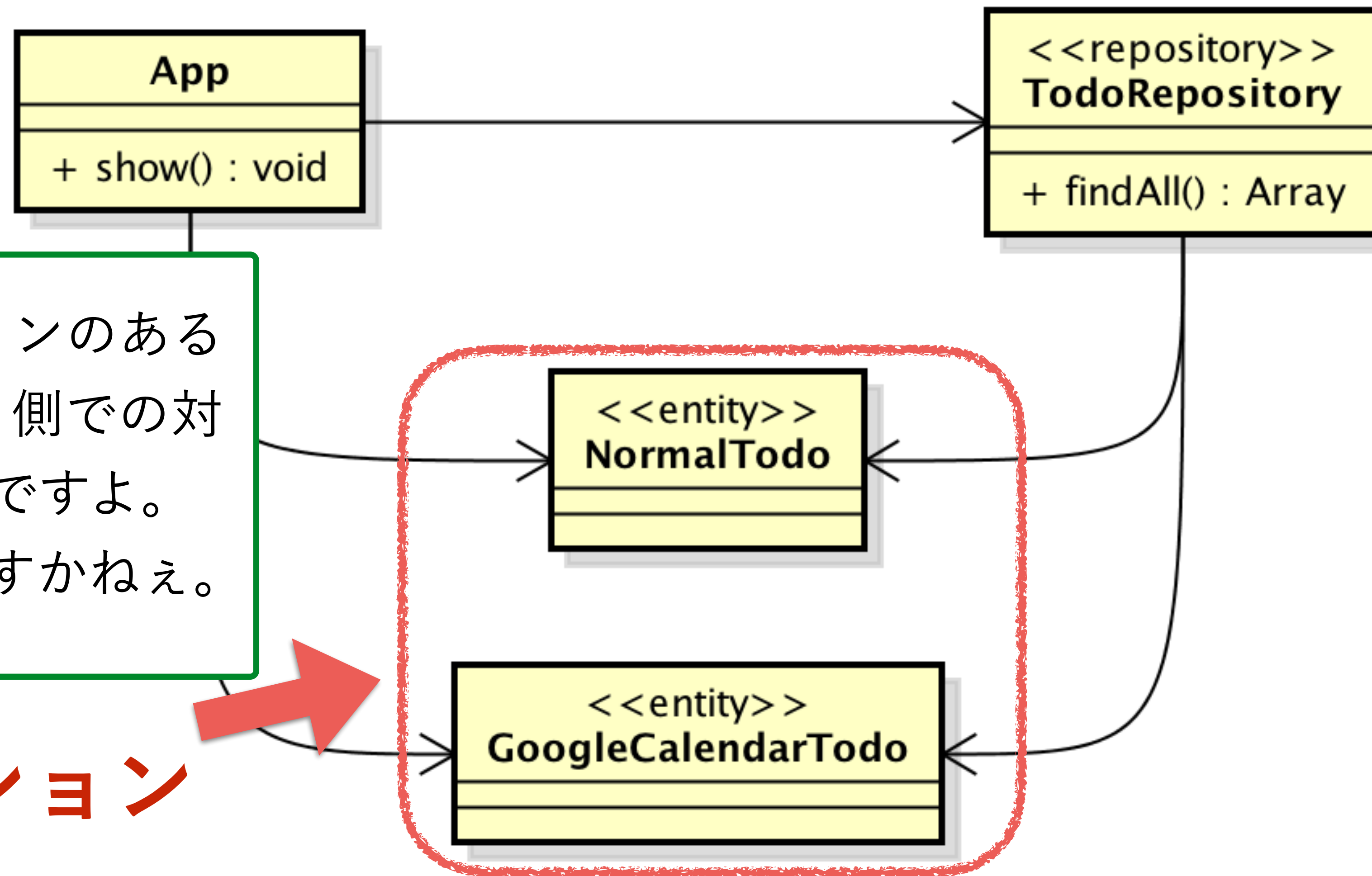


クラス的设计



バリエーション

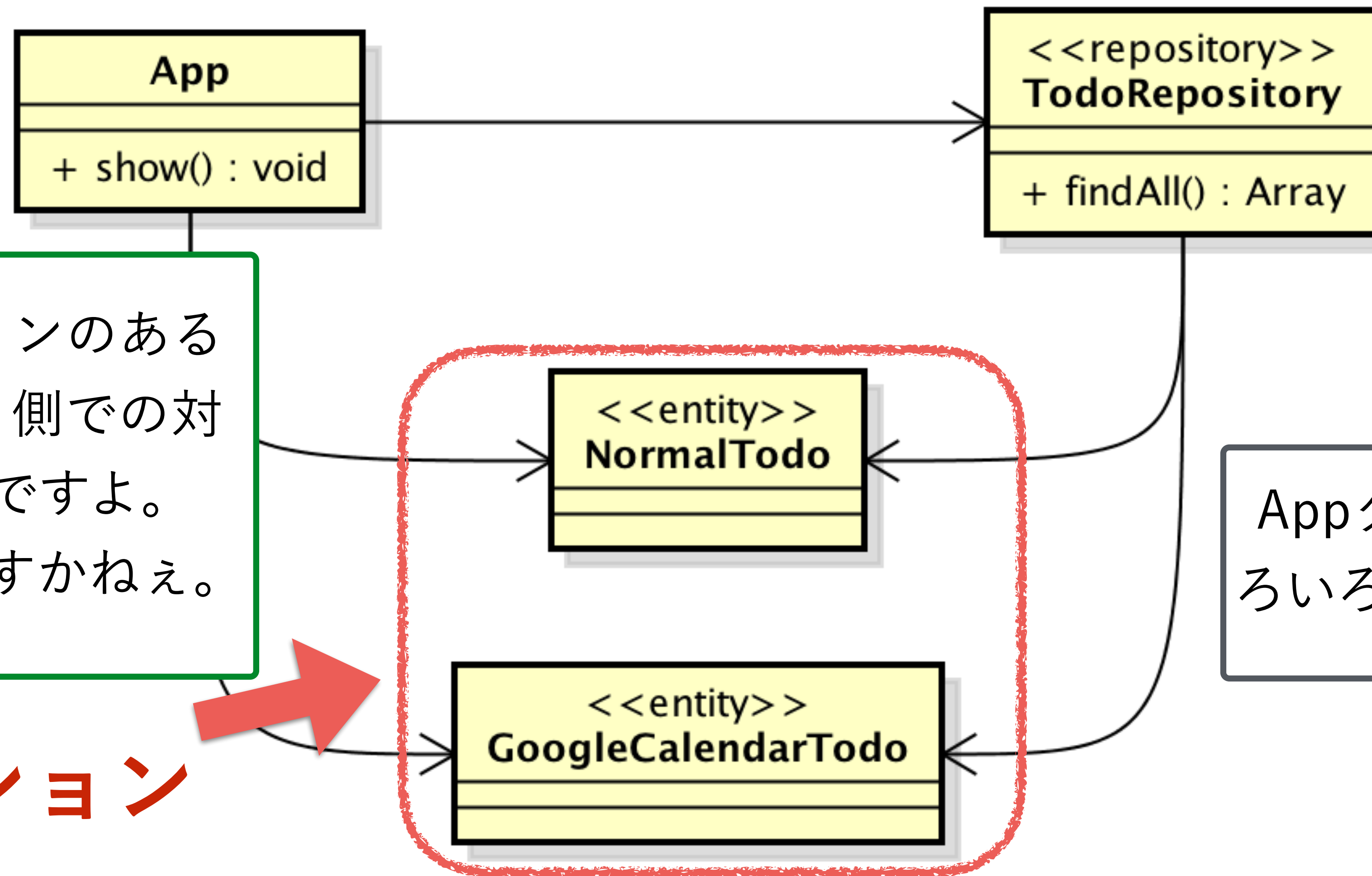
クラス的设计



バリエーションのある
データを使う側での対
応が肝なんですよ。
どうなってますかねえ。

バリエーション

クラス的设计



バリエーションのある
データを使う側での対
応が肝なんですよ。
どうなってますかねえ。

バリエーション

Appクラスの中でい
ろいろやっています！

Appクラスのshow()メソッド

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        echo '-----';  
        echo $this->formatTitle($todo) . PHP_EOL;  
        echo $todo->getUrl() . PHP_EOL;  
    }  
}
```

Appクラスのshow()メソッド

なるほど。こう分けたか。
表示の要件に気を取られちゃっ
て、バリエーションを上手く
扱えてなさそうですね。

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        echo $this->formatTitle($todo) . PHP_EOL;  
        echo $todo->getUrl() . PHP_EOL;  
    }  
}
```


Appクラスのshow()メソッド

なるほど。こう分けたか。
表示の要件に気を取られちゃっ
て、バリエーションを上手く
扱えてなさそうですね。

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        echo $this->formatTitle($todo) . PHP_EOL;  
        echo $todo->getUrl()  
    }  
}
```

上手く扱えてないというのは、
どういうことなんスか？

AppクラスのformatTitle()メソッド

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
    return $title;  
}
```

AppクラスのformatTitle()メソッド

例えばこのformatTitle()メソッド。1つのメソッドで2種類のTODOのタイトルを同時に扱っていますよね。

```
title($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
  
    return $title;  
}
```


AppクラスのformatTitle()メソッド

例えばこのformatTitle()メソッド。1つのメソッドで2種類のTODOのタイトルを同時に扱っていますよね。

はい、良い感じにまとめました！

```
title($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatusLabel($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
  
    return $title;  
}
```

AppクラスのformatTitle()メソッド

例えばこのformatTitle()メソッド。1つのメソッドで2種類のTODOのタイトルを同時に扱っていますよね。

もし3つ目のTODOが増えて、そのTODOは、タイトルの後にステータスを表示する仕様だったら、どうします？

はい、良い感じにまとめました！

```
title($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $this->formatStatusLabel($todo->getTitle());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $this->formatStatusOfGoogleCalendar($todo->getTitle());  
    }  
    return $title;  
}
```

AppクラスのformatTitle()メソッド

例えばこのformatTitle()メソッド。1つのメソッドで2種類のTODOのタイトルを同時に扱っていますよね。

もし3つ目のTODOが増えて、そのTODOは、タイトルの後にステータスを表示する仕様だったら、どうします？

はい、良い感じにまとめました！

う、めんどくさそうッスね・・・

```
title($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $this->formatStatusLabel($todo->getTitle());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $this->formatStatusOfGoogleCalendar($todo->getTitle());  
    }  
    return $title;  
}
```


常に未知の仕様を検討しろというわけじゃないですよ。検討すべき根拠のある部分をするんです。

常に未知の仕様を検討しろというわけじゃないですよ。検討すべき根拠のある部分をするんです。

そうなんですね。
安心しました。

常に未知の仕様を検討しろというわけじゃないですよ。検討すべき根拠のある部分をするんです。

今回は、TODOがすでに2種類ある状況です。これをたやすく3種類に増やせるようになっているかを考えることが、実はコード設計につながるんです。

そうなんですね。
安心しました。

常に未知の仕様を検討しろというわけじゃないですよ。検討すべき根拠のある部分をするんです。

今回は、TODOがすでに2種類ある状況です。これをたやすく3種類に増やせるようになっているかを考えることが、実はコード設計につながるんです。

そうなんですね。
安心しました。

そんな見方があるんですね！

常に未知の仕様を検討しろというわけじゃないですよ。検討すべき根拠のある部分をするんです。

今回は、TODOがすでに2種類ある状況です。これをたやすく3種類に増やせるようになっているかを考えることが、実はコード設計につながるんです。

「オープン・クローズドの原則」
というのがあるんですよ。

そうなんですね。
安心しました。

そんな見方があるんですね！

常に未知の仕様を検討しろというわけじゃないですよ。検討すべき根拠のある部分をするんです。

今回は、TODOがすでに2種類ある状況です。これをたやすく3種類に増やせるようになっているかを考えることが、実はコード設計につながるんです。

「オープン・クローズドの原則」
というのがあるんですよ。

そうなんですね。
安心しました。

そんな見方があるんですね！

聞いたことはありますが、しっかり理解できてないッス。
是非教えてください！！

オープン・クローズドの原則

オープン・クローズドの原則

オープン

機能を拡張できる

オープン・クローズドの原則

オープン

機能を拡張できる

クローズド

修正を行わない

オープン・クローズドの原則


オープン

機能を拡張できる

クローズド

修正を行わない

同時に満たすこと！！




オープン・クローズドの原則

オープン

機能を拡張できる

クローズド

修正を行わない



同時に満たすこと！！

モジュールに新たな振る舞いを追加する際に、
既存のコードを修正せず、単に新しいコードを追加するだけで
目的を達成できる状態になっていること。

オープン・クローズドの原則の適用範囲

オープン・クローズドの原則の適用範囲

モジュールに新たな振る舞いを追加する際に、
既存のコードを修正せず、単に新しいコードを追加するだけで
目的を達成できる状態になっていること。

オープン・クローズドの原則の適用範囲

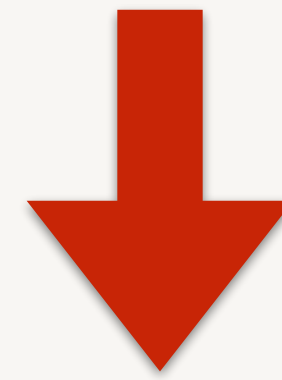
バリエーションを起因として

モジュールに新たな振る舞いを追加する際に、
既存のコードを修正せず、単に新しいコードを追加するだけで
目的を達成できる状態になっていること。

オープン・クローズドの原則の適用範囲

バリエーションを起因として

モジュールに新たな振る舞いを追加する際に、
既存のコードを修正せず、単に新しいコードを追加するだけで
目的を達成できる状態になっていること。



バリエーションからコードを保護せよ！！

オープン・クローズドの原則の着眼点

オープン・クローズドの原則の着眼点

バリエーションによって

変化する部分はどこか

オープン・クローズドの原則の着眼点

バリエーションによって

変化する部分はどこか

それらが、バリエーションの軸に沿って

まとめられているか

オープン・クローズドの原則の着眼点

バリエーションによって 変化する部分はどこか

それらが、バリエーションの軸に沿って まとめられているか

コードで見てみましょう。

バリエーションの軸に沿ったコードかどうか？

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $statusLabel = $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $statusLabel = $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
    return $title;  
}
```

バリエーションの軸に沿ったコードかどうか？

先程見たformatTitle()メソッド。2種類のTODOそれぞれの処理が入り組んでいますね。

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
  
    return $title;  
}
```


バリエーションの軸に沿ったコードかどうか？

先程見たformatTitle()メソッド。2種類のTODOそれぞれの処理が入り組んでいますね。

たしかに。

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $title = sprintf('%s%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $title .= sprintf(' (%s)', $todo->getLocation());  
    }  
  
    return $title;  
}
```

バリエーションの軸に沿ったコードかどうか？

先程見たformatTitle()メソッド。2種類のTODOそれぞれの処理が入り組んでいますね。

たしかに。

今回の要件ではTODOにバリエーションがあるので、それが軸です。TODOごとにコードがまとまっているといえるでしょうか？

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    $statusLabel = $this->formatStatus($todo->getStatus());  
    $title = sprintf('%s', $statusLabel, $todo->getTitle());  
  
    if ($todo instanceof GoogleCalendarTodo) {  
        $location = sprintf('%s (%s)', $title, $todo->getLocation());  
    }  
  
    return $title;  
}
```

バリエーションの軸に沿ったコードかどうか？

先程見たformatTitle()メソッド。2種類のTODOそれぞれの処理が入り組んでいますね。

たしかに。

今回の要件ではTODOにバリエーションがあるので、それが軸です。TODOごとにコードがまとまっているといえるでしょうか？

「行」「タイトル」というまとまりで考えていたので、
TODOごとにはなってないっスね・・・

```
private function formatTitle($todo): string {  
    if ($todo instanceof NormalTodo) {  
        $this->formatStatus($todo->getStatus());  
    } elseif ($todo instanceof GoogleCalendarTodo) {  
        $this->formatStatusOfGoogleCalendar($todo);  
    }  
  
    return $title;  
}
```


一覧の表示機能は、TODOの種類が1種類から2種類、2種類から3種類と増える時、つまりバリエーションの増加に対して、なんらかの拡張を行う必要がありますよね。

一覧の表示機能は、TODOの種類が1種類から2種類、2種類から3種類と増える時、つまりバリエーションの増加に対して、なんらかの拡張を行う必要がありますよね。

はい。拡張が必要です。

一覧の表示機能は、TODOの種類が1種類から2種類、2種類から3種類と増える時、つまりバリエーションの増加に対して、なんらかの拡張を行う必要がありますよね。

しかし、そのためには、既存のコードのあちこちを修正しなくてはならない状況ですよね。

はい。拡張が必要です。

一覧の表示機能は、TODOの種類が1種類から2種類、2種類から3種類と増える時、つまりバリエーションの増加に対して、なんらかの拡張を行う必要がありますよね。

しかし、そのためには、既存のコードのあちこちを修正しなくてはならない状況ですよね。

はい。拡張が必要です。

それしかないように思えます。

一覧の表示機能は、TODOの種類が1種類から2種類、2種類から3種類と増える時、つまりバリエーションの増加に対して、なんらかの拡張を行う必要がありますよね。

しかし、そのためには、既存のコードのあちこちを修正しなくてはならない状況ですよね。

この状況を、オープン・クローズドの原則に違反している、というんです。

はい。拡張が必要です。

それしかないように思えます。

一覧の表示機能は、TODOの種類が1種類から2種類、2種類から3種類と増える時、つまりバリエーションの増加に対して、なんらかの拡張を行う必要がありますよね。

しかし、そのためには、既存のコードのあちこちを修正しなくてはならない状況ですよ。

この状況を、オープン・クローズドの原則に違反している、というんです。

はい。拡張が必要です。

それしかないように思えます。

おっ、OCP違反
というやつッスね！
(言ってみたかった)

オープン・クローズドの原則に
準拠させてみよう！

オープン・クローズドの原則に 準拠させてみよう！

一緒にやってみましょうか！

オープン・クローズドの原則に 準拠させてみよう！

一緒にやってみましょうか！

オナシャッス！！

何かから着手したらよいのか？

何から着手したらよいのか？

バリエーションの軸を捉え直し

何から着手したらよいのか？

バリエーションの軸を捉え直し



わりと大きなコード修正になる

何から着手したらよいのか？

バリエーションの軸を捉え直し



わりと大きなコード修正になる

だからバリエーションの軸の見極めが大事なんスね！

Step 1: Todoの種類ごとの処理をまとめなおす

```
class App {  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case NormalTodo::class:  
                    $this->showNormalTodo($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->showGoogleCalendarTodo($todo);  
                    break;  
            }  
        }  
    }  
  
    private function showNormalTodo(NormalTodo $todo) { /* ... */ }  
    private function showGoogleCalendarTodo(GoogleCalendarTodo $todo) { /* ... */ }  
}
```

Step 1: Todoの種類ごとの処理をまとめなおす

こんな風にswitchでTodoごとの処理を振り分けます。

```
    p {  
        private function show() {  
            $allTodo = $this->todoRepository->findAll();  
            foreach ($allTodo as $todo) {  
                switch (\get_class($todo)) {  
                    case NormalTodo::class:  
                        $this->showNormalTodo($todo);  
                        break;  
                    case GoogleCalendarTodo::class:  
                        $this->showGoogleCalendarTodo($todo);  
                        break;  
                }  
            }  
        }  
    }  
    private function showNormalTodo(NormalTodo $todo) { /* ... */ }  
    private function showGoogleCalendarTodo(GoogleCalendarTodo $todo) { /* ... */ }  
}
```


Step 1: Todoの種類ごとの処理をまとめなおす

こんな風にswitchでTodoごとの処理を振り分けます。

```
public function show() {  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        switch (\get_class($todo)) {  
            case NormalTodo::class:  
                $this->showNormalTodo($todo);  
                break;  
            case GoogleCalendarTodo::class:  
                $this->showGoogleCalendarTodo($todo);  
                break;  
        }  
    }  
}  
  
private function showNormalTodo(NormalTodo $todo) { /* ... */ }  
private function showGoogleCalendarTodo(GoogleCalendarTodo $todo) { /* ... */ }  
}
```

なるほど、一番おおもとのところで、種類ごとに分けちゃうんですね。

Step 1: Todoの種類ごとの処理をまとめなおす

こんな風にswitchでTodoごとの処理を振り分けます。

そうです。それが、「バリエーションの軸に沿っている」ということなんです。

なるほど、一番おおもとのところで、種類ごとに分けちゃうスね。

```
public function show() {
    $allTodo = $this->todoRepository->findAll();
    foreach ($allTodo as $todo) {
        switch (\get_class($todo)) {
            case NormalTodo::class:
                $this->showNormalTodo($todo);
                break;
            case GoogleCalendarTodo::class:
                $this->showGoogleCalendarTodo($todo);
                break;
        }
    }
}

private function showNormalTodo(NormalTodo $todo) { /* ... */ }
private function showGoogleCalendarTodo(GoogleCalendarTodo $todo) { /* ... */ }
}
```

Step 1: Todoの種類ごとの処理をまとめなおす

```
class App {  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case NormalTodo::class:  
                    $this->showNormalTodo($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->showGoogleCalendarTodo($todo);  
                    break;  
            }  
        }  
    }  
  
    private function showNormalTodo(NormalTodo $todo) { /* ... */ }  
    private function showGoogleCalendarTodo(GoogleCalendarTodo $todo) { /* ... */ }  
}
```

表示処理本体は一旦プライベートメソッドにしておいて、次のステップでクラスにしましょう。

Step 2:表示処理をPresenterクラスへ分割(1)

```
class NormalTodoPresenter
{
    public function present(NormalTodo $todo)
    {
        // :
    }
}

class GoogleCalendarTodoPresenter
{
    public function present(GoogleCalendarTodo $todo)
    {
        // :
    }
}
```

Step 2:表示処理をPresenterクラスへ分割(1)

```
class NormalTodoPresenter
{
    public function present(NormalTodo $todo)
    {
```

Appクラスのプライベートメソッドにしておいた表示処理を、クラスに分割しました。それぞれpresent()メソッドで表示処理を行います。

```
GoogleCalendarTodoPresenter
```

```
function present(GoogleCalendarTodo $todo)
```

```

    }
}
}
```


Step 2:表示処理をPresenterクラスへ分割(1)

```
class NormalTodoPresenter
{
    public function present(NormalTodo $todo)
    {
```

Appクラスのプライベートメソッドにしておいた表示処理を、クラスに分割しました。それぞれpresent()メソッドで表示処理を行います。

```
GoogleCalendarTodoPresenter
```

```
function present(GoogleCalendarTodo $todo)
```

なるほどなるほど

Step 2:表示処理をPresenterクラスへ分割(2)

Appクラスでは、先程作ったPresenterのインスタンスを使うだけにします。Todoごとの表示処理を外に追い出せたので、Appクラスはだいぶスッキリしましたね。

```
class App {  
    public function __construct(  
        TodoRepository $todoRepository,  
        TodoPresenter $todoPresenter,  
        GoogleCalendarTodoPresenter $googleCalendarTodoPresenter  
    ) {  
        $this->todoRepository = $todoRepository;  
        // ...  
    }  
  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case Todo::class:  
                    $this->todoPresenter->present($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->googleCalendarTodoPresenter->present($todo);  
                    break;  
            }  
        }  
    }  
}
```


Step 2:表示処理をPresenterクラスへ分割(2)

Appクラスでは、先程作ったPresenterのインスタンスを使うだけにします。

Todoごとの表示処理を外に追い出せたので、Appクラスはだいぶスッキリしましたね。

たしかに！

```

class App {
    public function __construct(
        TodoRepository $todoRepository,
        TodoPresenter $todoPresenter,
        GoogleCalendarTodoPresenter $googleCalendarTodoPresenter
    ) {
        $this->todoRepository = $todoRepository;
        // ...
    }

    public function show() {
        $allTodo = $this->todoRepository->findAll();
        foreach ($allTodo as $todo) {
            switch (\get_class($todo)) {
                case Todo::class:
                    $this->todoPresenter->present($todo);
                    break;
                case GoogleCalendarTodo::class:
                    $this->googleCalendarTodoPresenter->present($todo);
                    break;
            }
        }
    }
}

```

たしかに

Step 2:表示処理をPresenterクラスへ分割(2)

ところで、この状態で、オープン・クローズドの原則に準拠するようになったと思いますか？

```
class App {  
    public function __construct(  
        TodoRepository $todoRepository,  
        TodoPresenter $todoPresenter,  
        GoogleCalendarTodoPresenter $googleCalendarTodoPresenter  
    ) {  
        $this->todoRepository = $todoRepository;  
        // ...  
    }  
  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case Todo::class:  
                    $this->todoPresenter->present($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->googleCalendarTodoPresenter->present($todo);  
                    break;  
            }  
        }  
    }  
}
```


Step 2:表示処理をPresenterクラスへ分割(2)

```
class App {  
    public function __construct(  
        TodoRepository $todoRepository,  
        TodoPresenter $todoPresenter,  
        GoogleCalendarTodoPresenter $googleCalendarTodoPresenter  
    ) {  
        $this->todoRepository = $todoRepository;  
        // ...  
    }  
  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case Todo::class:  
                    $this->todoPresenter->present($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->googleCalendarTodoPresenter->present($todo);  
                    break;  
            }  
        }  
    }  
}
```

ところで、この状態で、オープン・クローズドの原則に準拠するようになったと思いますか？

うーん。
いろいろ分割したので、OK、にはなってないん
スか？

Step 2: 表示処理をPresenterクラスへ分割(2)

```
class App {  
    public function __construct(  
        TodoRepository $todoRepository,  
        TodoPresenter $todoPresenter,  
        GoogleCalendarTodoPresenter $googleCalendarTodoPresenter  
    ) {  
        $this->todoRepository = $todoRepository;  
        // ...  
    }  
  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case Todo::class:  
                    $this->todoPresenter->present($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->googleCalendarTodoPresenter->present($todo);  
                    break;  
            }  
        }  
    }  
}
```

ところで、この状態で、オープン・クローズドの原則に準拠するようになったと思いますか？

3種類目のTODOが増えた時、このswitchはどうなりますかね？

うーん。
いろいろ分割したので、OK、にはなってないんですか？

Step 2:表示処理をPresenterクラスへ分割(2)

```
class App {  
    public function __construct(  
        TodoRepository $todoRepository,  
        TodoPresenter $todoPresenter,  
        GoogleCalendarTodoPresenter $googleCalendarTodoPresenter  
    ) {  
        $this->todoRepository = $todoRepository;  
        // ...  
    }  
  
    public function show() {  
        $allTodo = $this->todoRepository->findAll();  
        foreach ($allTodo as $todo) {  
            switch (\get_class($todo)) {  
                case Todo::class:  
                    $this->todoPresenter->present($todo);  
                    break;  
                case GoogleCalendarTodo::class:  
                    $this->googleCalendarTodoPresenter->present($todo);  
                    break;  
            }  
        }  
    }  
}
```

ところで、この状態で、オープン・クローズドの原則に準拠するようになったと思いますか？

3種類目のTODOが増えた時、このswitchはどうなりますかね？

うーん。
いろいろ分割したので、OK、にはなってないん
スか？

あ”ーっ。

オープン・クローズドの原則には
まだ準拠していませんね。

オープン・クローズドの原則には
まだ準拠していませんね。

修正が必要なんですよ。

オープン・クローズドの原則には
まだ準拠していませんね。

修正が必要なんですよ。

たしかに。

オープン・クローズドの原則には まだ準拠していませんね。

修正が必要なんですよ。

準拠するように、最後までやりきってみましょう。もうひと踏ん張りして、switch自体を消し去ります！

たしかに。

オープン・クローズドの原則には まだ準拠していませんね。

修正が必要なんですよ。

準拠するように、最後までやりきってみましょう。もうひと踏ん張りして、switch自体を消し去ります！

switch文でやっていた仕事を、Resolverというクラスに任せます。

たしかに。

オープン・クローズドの原則には まだ準拠していませんね。

修正が必要なんですよ。

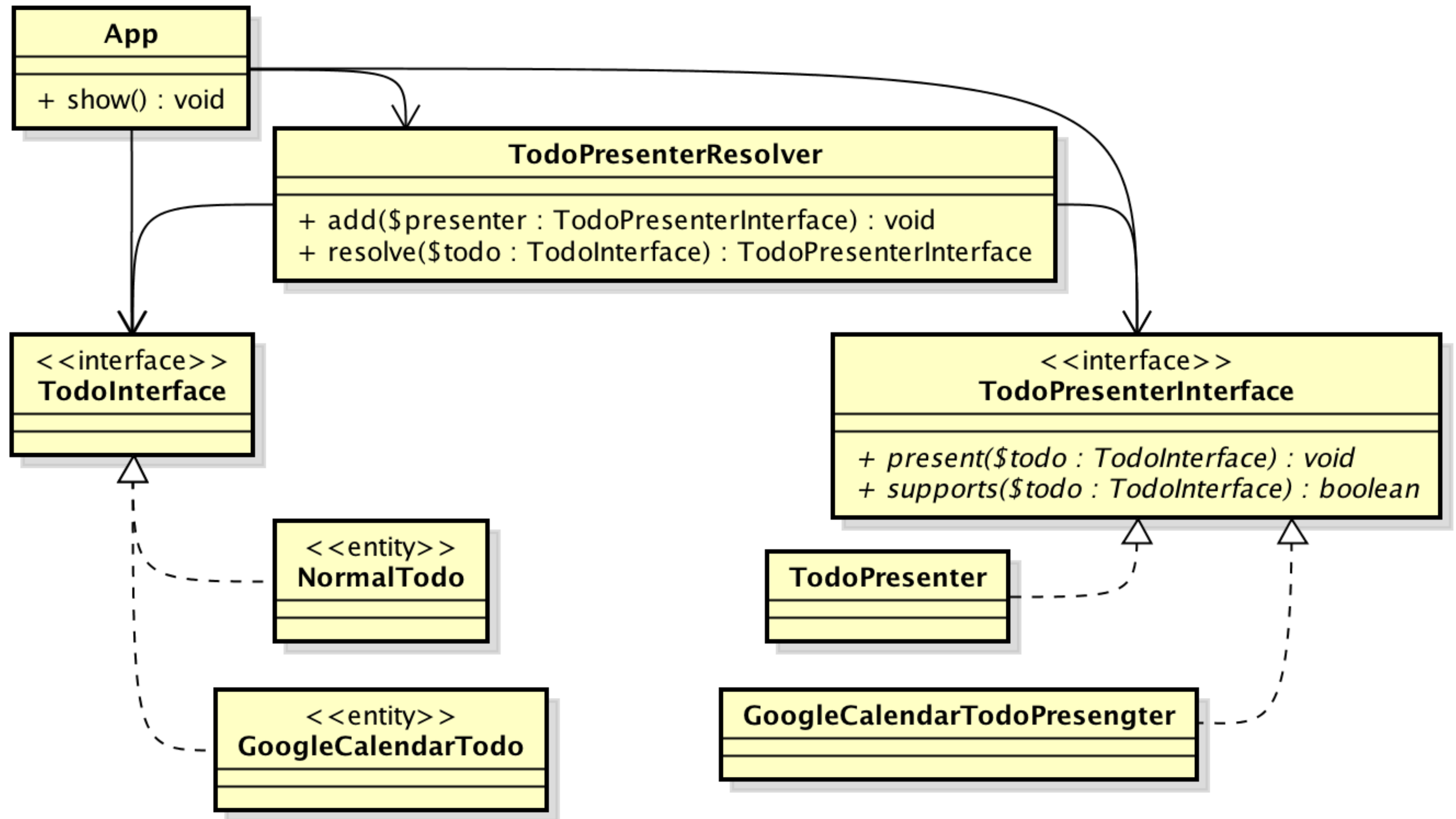
準拠するように、最後までやりきってみましょう。もうひと踏ん張りして、switch自体を消し去ります！

switch文でやっていた仕事を、Resolverというクラスに任せます。

たしかに。

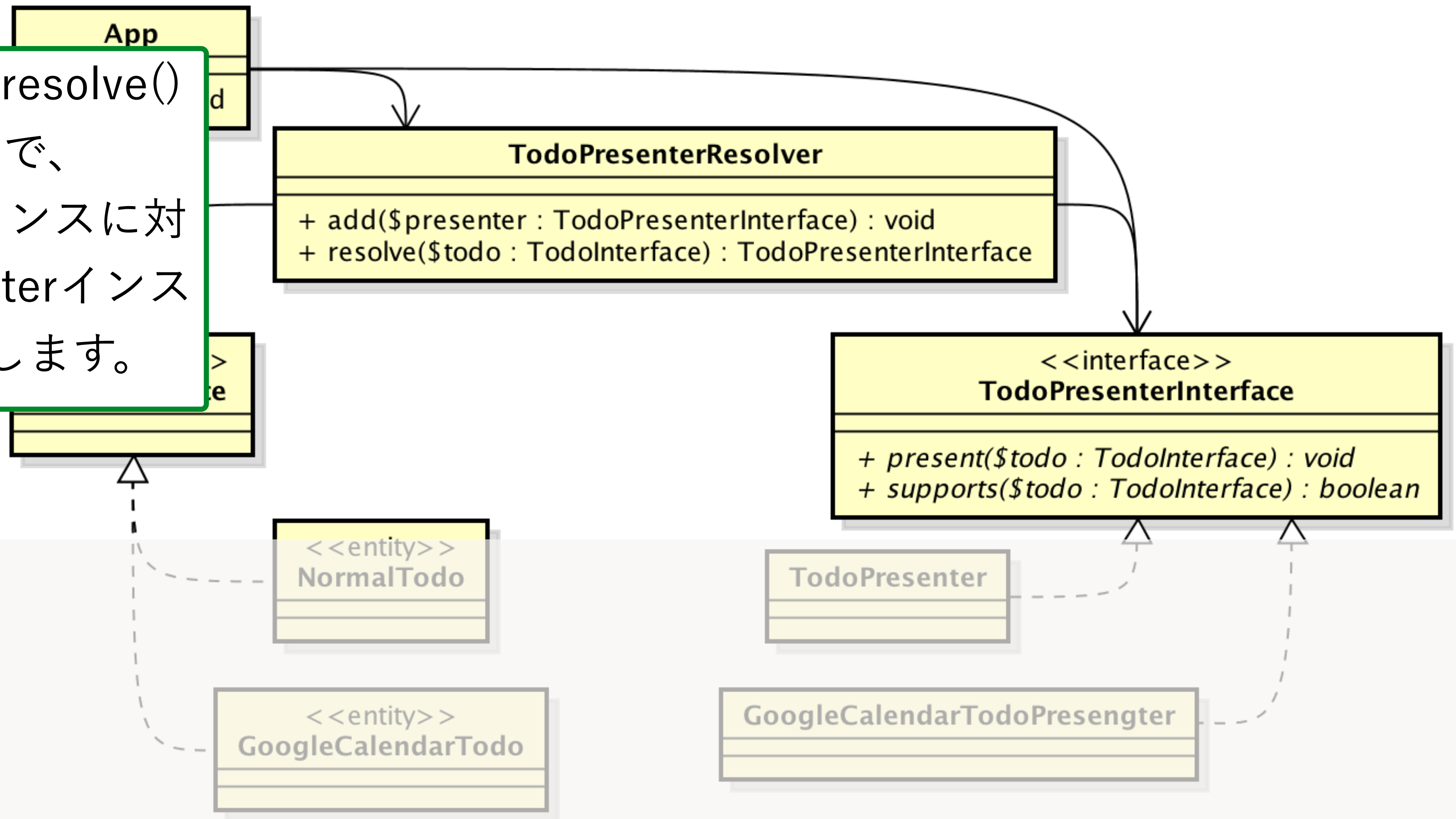
解決っていう仕事を抜き出すんスね。

Resolverを使う場合の全体像



Resolverを使う場合の全体像

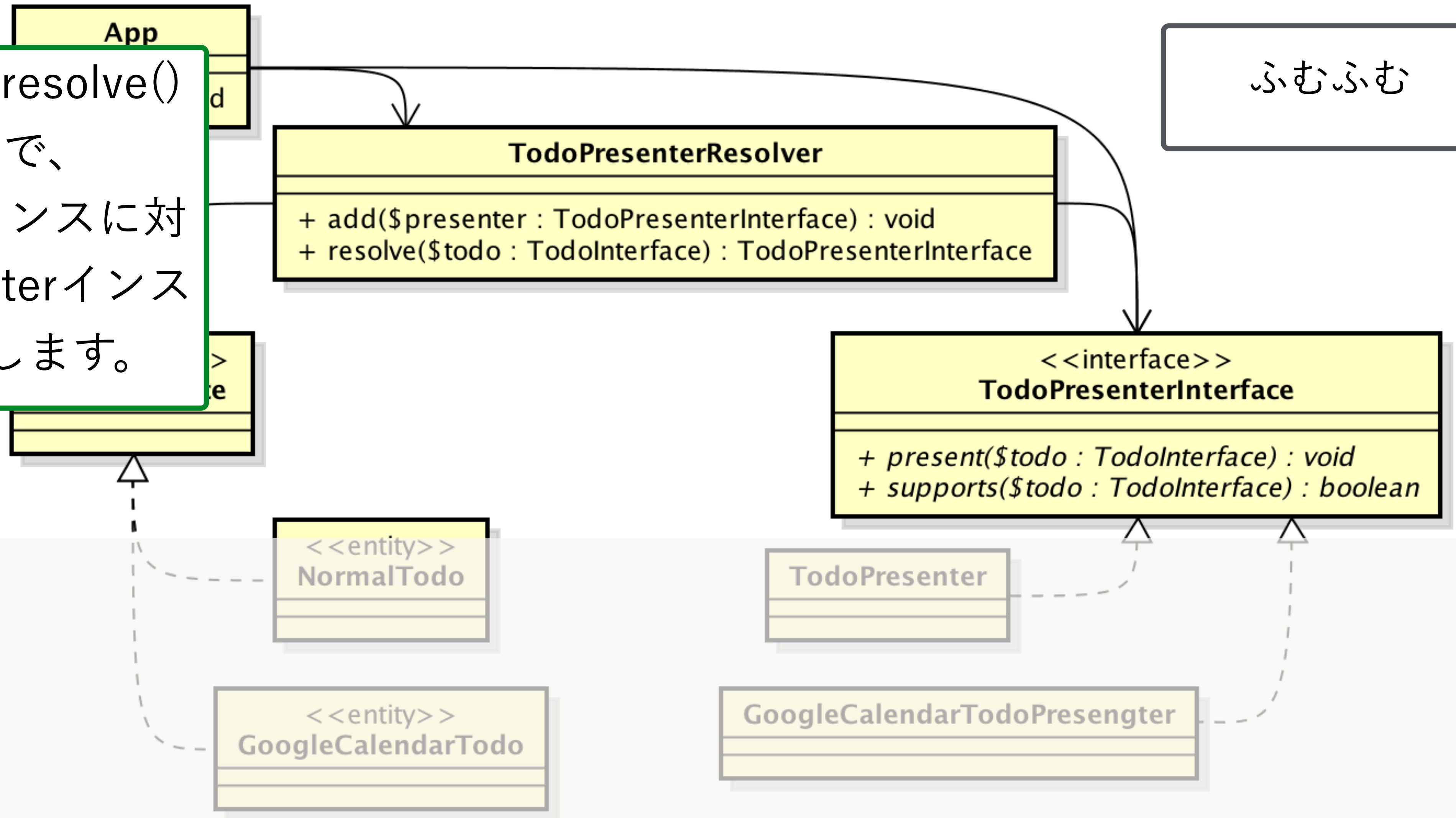
Resolverは、`resolve()`メソッドで、
Todoインスタンスに対応するPresenterインスタンスを返します。



Resolverを使う場合の全体像

Resolverは、resolve()
メソッドで、
Todoインスタンスに対
応するPresenterインス
タンスを返します。

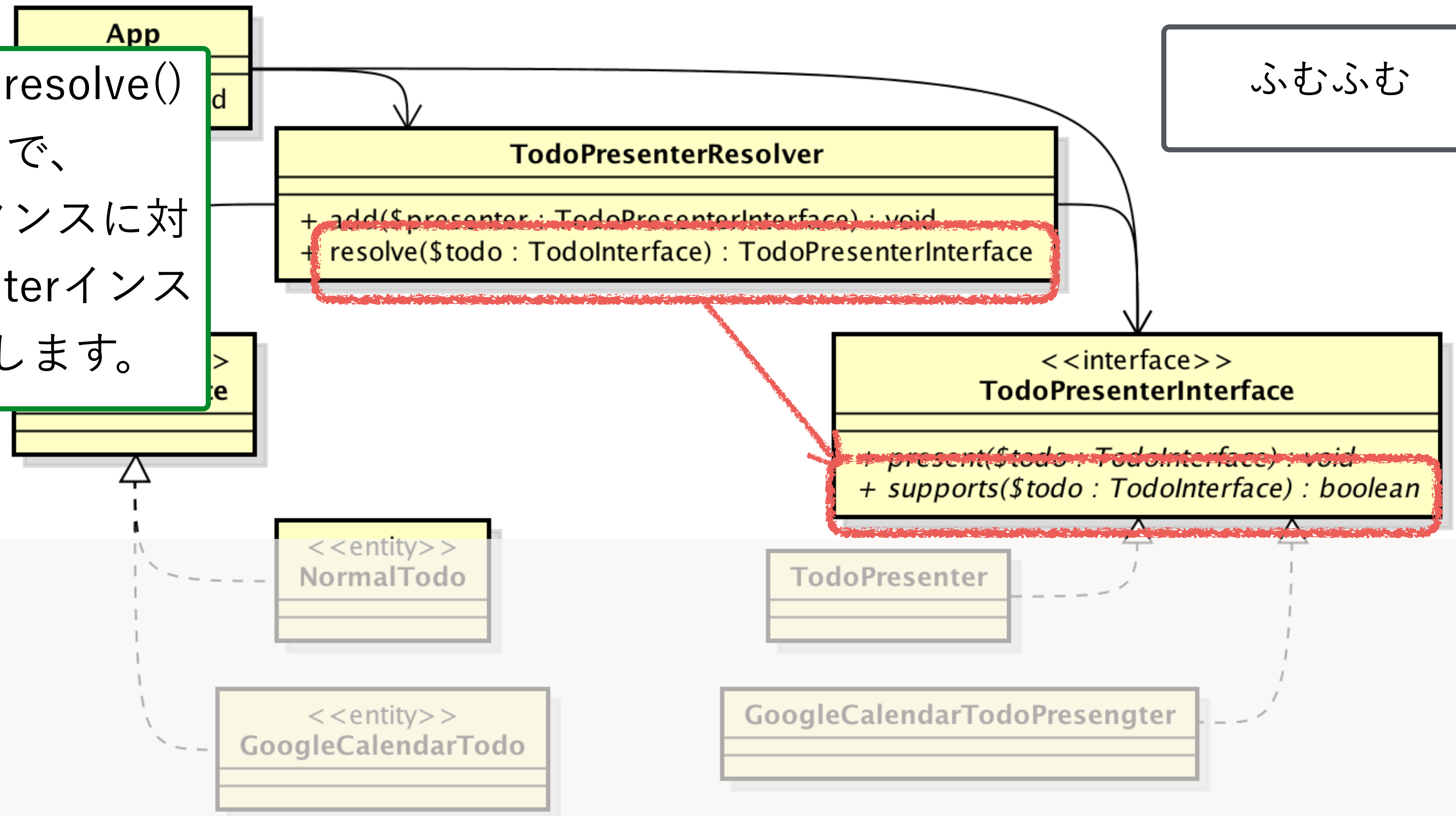
ふむふむ



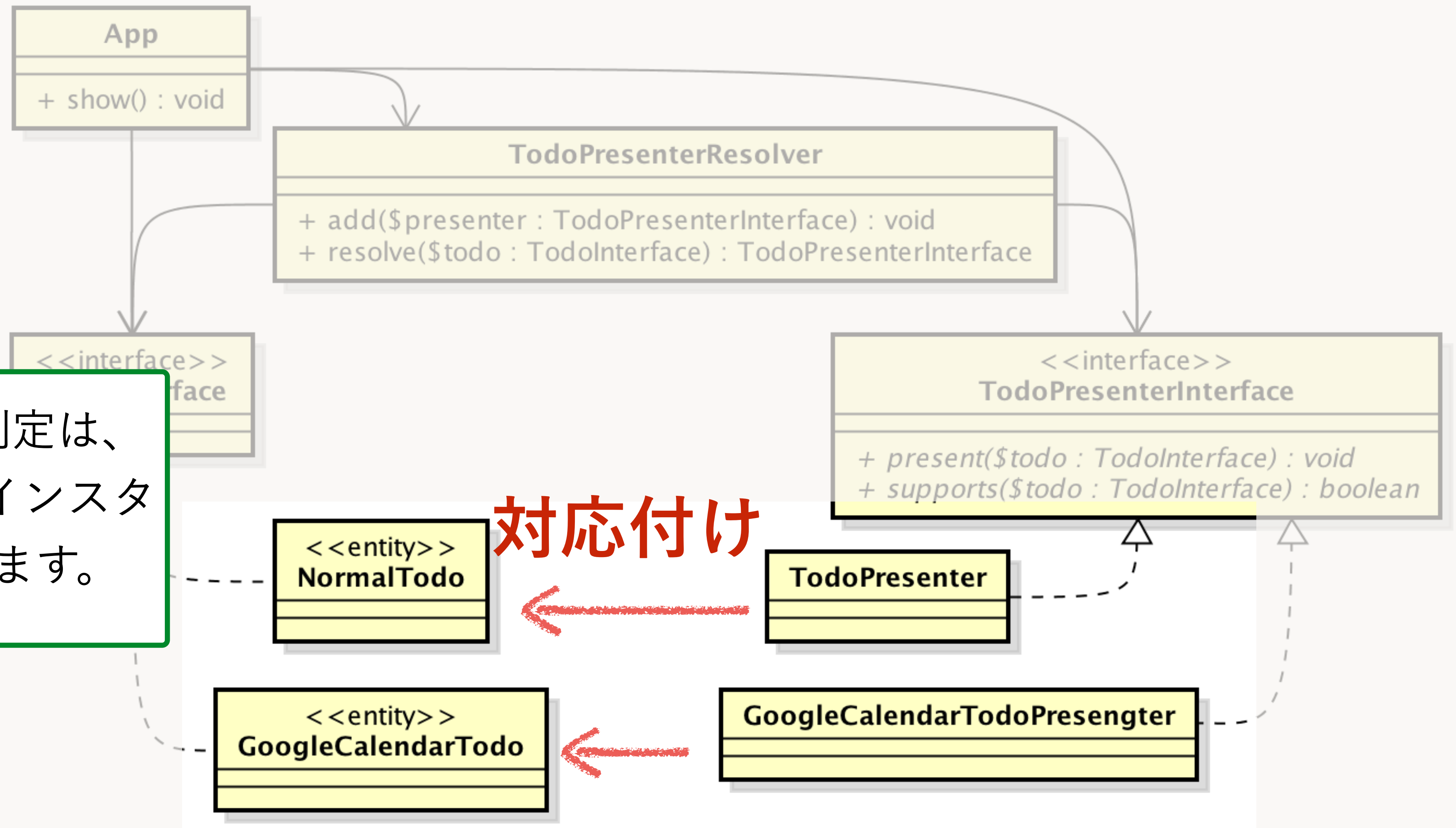
Resolverを使う場合の全体像

Resolverは、resolve()
メソッドで、
Todoインスタンスに対
応するPresenterインス
タンスを返します。

ふむふむ



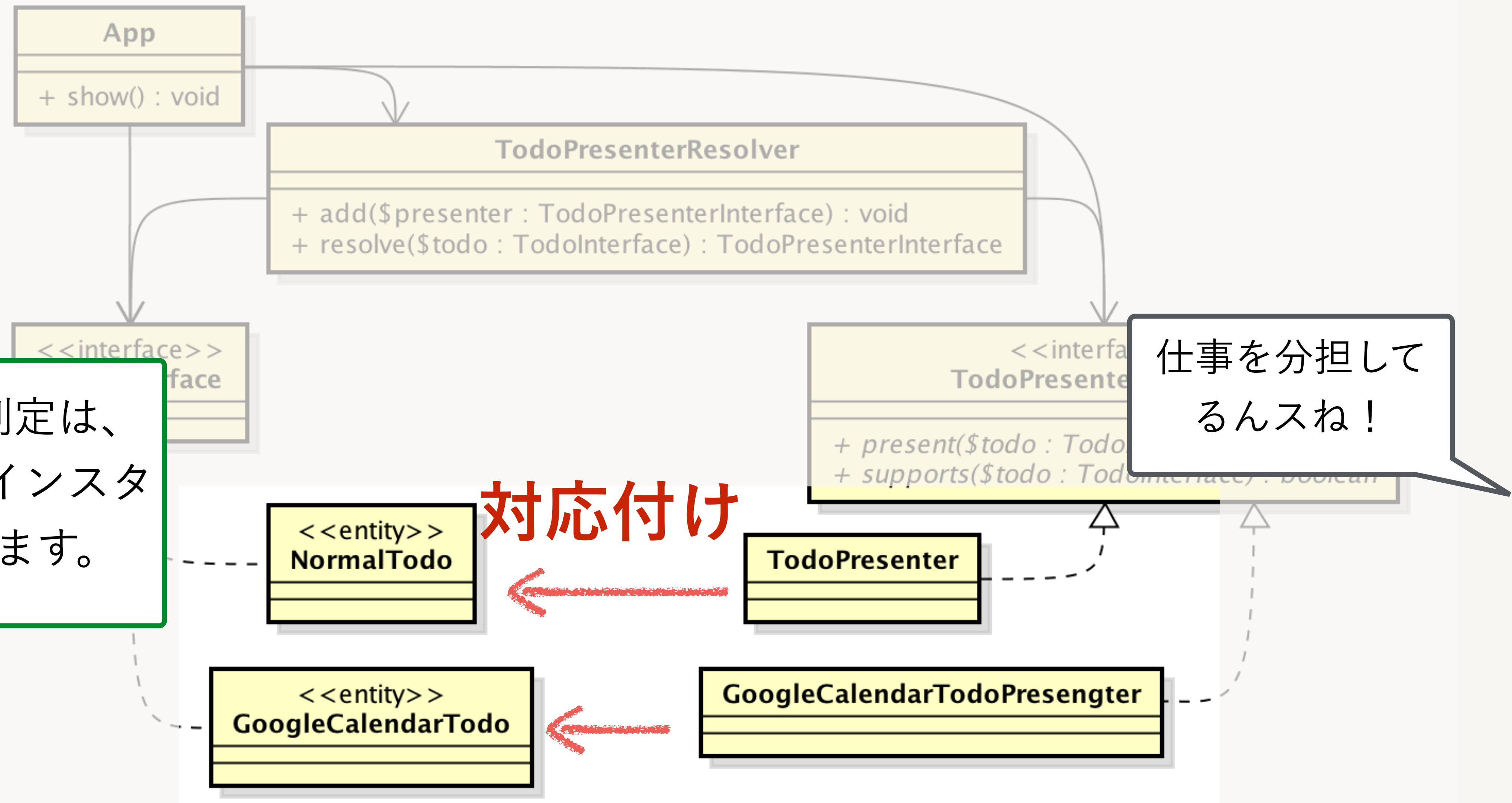
Resolverを使う場合の全体像



対応関係の判定は、
各Presenterインスタ
ンスに任せます。

対応付け

Resolverを使う場合の全体像



対応関係の判定は、
各Presenterインスタ
ンスに任せます。

仕事を分担して
るんスね！

対応付け

Step 3: Presenterクラスの修正

```
interface TodoPresenterInterface
{
    public function present(TodoInterface $todo);

    public function supports(TodoInterface $todo): bool;
}

class NormalTodoPresenter implements TodoPresenterInterface
{
    public function present(TodoInterface $todo) {
        /** @var NormalTodo $todo */
        // :
    }

    public function supports(TodoInterface $todo): bool {
        return $todo instanceof NormalTodo;
    }
}
```

Step 3: Presenterクラスの修正

TodoPresenterInterface
によって、Presenterク
ラスがpresent()メソッ
ドとsupports()メソッド
を持つことを保証します。

```
interface TodoPresenterInterface {  
    public function present(TodoInterface $todo);  
  
    public function supports(TodoInterface $todo): bool;  
}  
  
class NormalTodoPresenter implements TodoPresenterInterface  
{  
    public function present(TodoInterface $todo) {  
        /** @var NormalTodo $todo */  
        // :  
    }  
  
    public function supports(TodoInterface $todo): bool {  
        return $todo instanceof NormalTodo;  
    }  
}
```


Step 3: Presenterクラスの修正

TodoPresenterInterface
によって、Presenterク
ラスがpresent()メソッ
ドとsupports()メソッド
を持つことを保証します。

こういう時にインター
フェイスを使うんで
すね！

```
interface TodoPresenterInterface
```

```
public function present(TodoInterface $todo);
```

```
public function supports(TodoInterface $todo): bool;
```

```
class NormalTodoPresenter implements TodoPresenterInterface
```

```
public function present(TodoInterface $todo) {
```

```
    /** @var NormalTodo $todo */
```

```
    // :
```

```
}
```

```
public function supports(TodoInterface $todo): bool {
```

```
    return $todo instanceof NormalTodo;
```

```
}
```

```
}
```

Step 3: Presenterクラスの修正

Presenterクラスの
supports()メソッドで、
対応付け判定を行うよ
うにしておきます。
NormalTodo用の
Presenterなので、
NormalTodoならtrueを
返します。

```
interface TodoPresenterInterface
{
    public function present(TodoInterface $todo);

    public function supports(TodoInterface $todo): bool;
}

class NormalTodoPresenter implements TodoPresenterInterface
{
    public function present(TodoInterface $todo) {
        /** @var NormalTodo $todo */
        // :
    }

    public function supports(TodoInterface $todo): bool {
        return $todo instanceof NormalTodo;
    }
}
```

Step 3: Presenterクラスの修正

Presenterクラスの
supports()メソッドで、
対応付け判定を行うよ
うにしておきます。
NormalTodo用の
Presenterなので、
NormalTodoならtrueを
返します。

```
interface TodoPresenterInterface
{
    public function present(TodoInterface $todo);

    public function supports(TodoInterface $todo): bool;
}

class NormalTodoPresenter implements TodoPresenterInterface
{
    public function present(TodoInterface $todo): void
    {
        /** @var NormalTodo $todo */
        // :
    }

    public function supports(TodoInterface $todo): bool {
        return $todo instanceof NormalTodo;
    }
}
```

TodoクラスとPresenter
クラスが1対1に対応し
ているので、こう判定
すればOKなんスね。

Step 4: Resolver クラスの実装

```
class TodoPresenterResolver {  
    private $presenters = [];  
  
    public function add(TodoPresenterInterface $presenter) {  
        $this->presenters[] = $presenter;  
    }  
  
    public function resolve(TodoInterface $todo) {  
        foreach ($this->presenters as $presenter) {  
            if ($presenter->supports($todo)) {  
                return $presenter;  
            }  
        }  
        throw new \LogicException(  
            'Unsupported type: ' . \get_class($todo));  
    }  
}
```

Step 4: Resolver クラスの実装

Resolver クラスでは、add() メソッドで、Presenter のインスタンスを登録できるようにしておきます。

```
class TodoPresenterResolver {  
    private $presenters = [];  
  
    public function add(TodoPresenterInterface $presenter) {  
        $this->presenters[] = $presenter;  
    }  
  
    public function resolve(TodoInterface $todo) {  
        foreach ($this->presenters as $presenter) {  
            if ($presenter->supports($todo)) {  
                return $presenter;  
            }  
        }  
        throw new \LogicException(  
            'Unsupported type: ' . \get_class($todo));  
    }  
}
```

Step 4: Resolver クラスの実装

resolve()メソッドは、引数で受け取ったTodoインスタンスに対応するPresenterインスタンスを解決して返します。

```
class TodoPresenterResolver {  
    private $presenters = [];  
  
    public function add(TodoPresenterInterface $presenter) {  
        $this->presenters[] = $presenter;  
    }  
  
    public function resolve(TodoInterface $todo) {  
        foreach ($this->presenters as $presenter) {  
            if ($presenter->supports($todo)) {  
                return $presenter;  
            }  
        }  
  
        throw new \LogicException(  
            'Unsupported type: ' . \get_class($todo));  
    }  
}
```


Step 4: Resolverクラスの実装

resolve()メソッドは、引数で受け取ったTodoインスタンスに対応するPresenterインスタンスを解決して返します。

登録されたPresenterインスタンスのsupports()メソッドを次々に呼び出して、対応するかどうか調べています。

```
class TodoPresenterResolver {  
    private $presenters = [];  
  
    public function add(TodoPresenterInterface $presenter) {  
        $this->presenters[] = $presenter;  
    }  
  
    public function resolve(TodoInterface $todo) {  
        foreach ($this->presenters as $presenter) {  
            if ($presenter->supports($todo)) {  
                return $presenter;  
            }  
        }  
  
        throw new \LogicException(  
            'Unsupported type: ' . \get_class($todo));  
    }  
}
```

Step 4: Resolverクラスの実装

resolve()メソッドは、引数で受け取ったTodoインスタンスに対応するPresenterインスタンスを解決して返します。

登録されたPresenterインスタンスのsupports()メソッドを次々に呼び出して、対応するかどうか調べています。

```
class TodoPresenterResolver {  
    private $presenters = [];  
  
    public function add(TodoPresenterInterface $presenter) {  
        $this->presenters[] = $presenter;  
    }  
  
    public function resolve(TodoInterface $todo) {  
        foreach ($this->presenters as $presenter) {  
            if ($presenter->supports($todo)) {  
                return $presenter;  
            }  
        }  
  
        throw new \LogicException('Unsupported type');  
    }  
}
```

なるほどー。ResolverにはTodo個別のコードは全く出てこないんですね。

Step 5: Appクラスの修正

```
class App
{
    public function __construct(
        TodoRepository $todoRepository,
        TodoPresenterResolver $presenterResolver
    ) {
        $this->todoRepository = $todoRepository;
        $this->presenterResolver = $presenterResolver;
    }

    public function show()
    {
        $allTodo = $this->todoRepository->findAll();
        foreach ($allTodo as $todo) {
            $presenter = $this->presenterResolver->resolve($todo);
            $presenter->present($todo);
        }
    }
}
```


Step 5: Appクラスの修正

最終的にAppクラスのshow()メソッドでは、ResolverからPresenterを受け取り、present()メソッドを呼び出します。

たったこれだけになりました！

```
class App
```

```
construct(  
    $todoRepository,  
    $resolver $presenterResolver  
) {  
    $this->todoRepository = $todoRepository;  
    $this->presenterResolver = $presenterResolver;  
}
```

```
public function show()  
{  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        $presenter = $this->presenterResolver->resolve($todo);  
        $presenter->present($todo);  
    }  
}
```


Step 5: Appクラスの修正

```
class App
```

最終的にAppクラスのshow()メソッドでは、ResolverからPresenterを受け取り、present()メソッドを呼び出だけです。

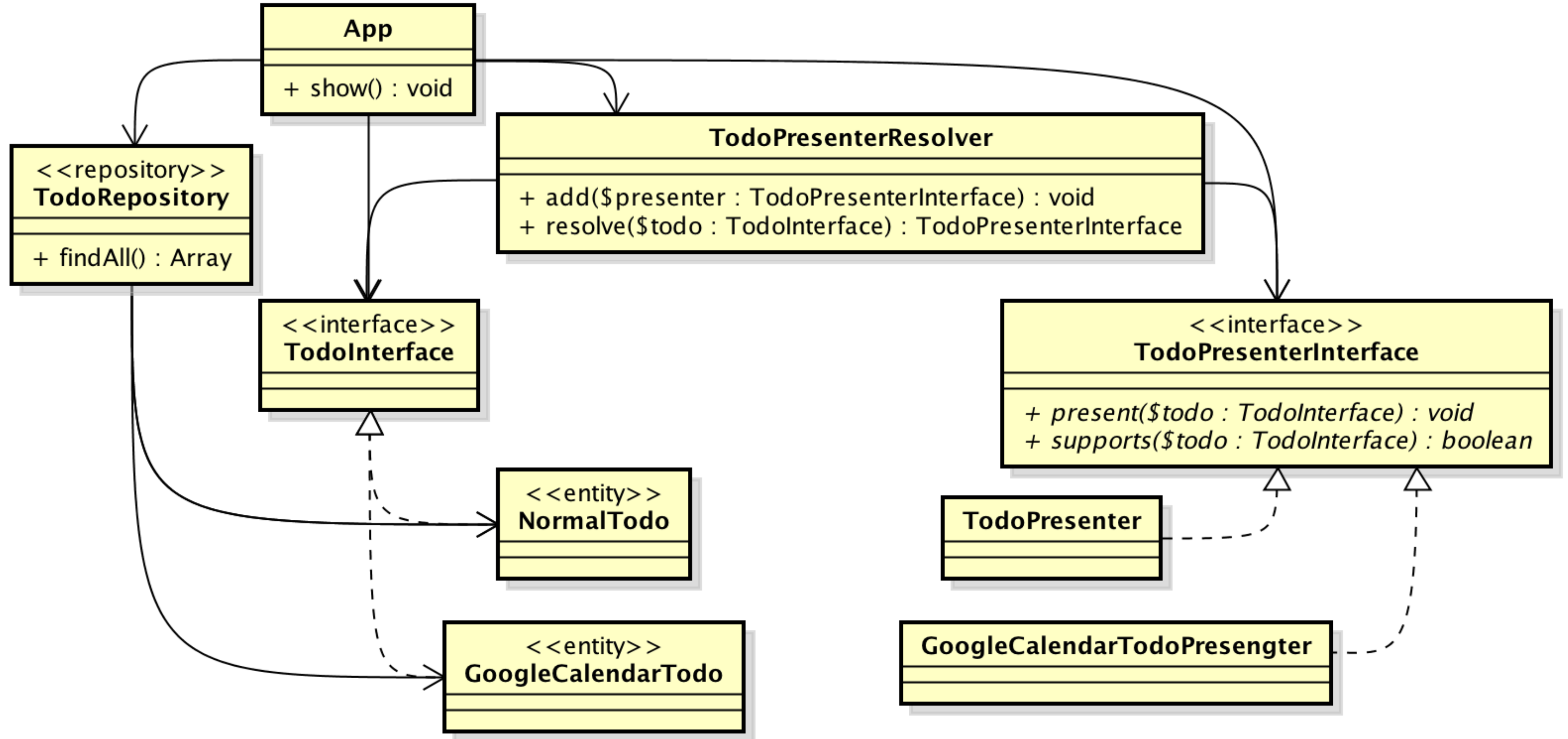
たったこれだけになりました！

```
construct(  
    $todoRepository,  
    $resolver $presenterResolver  
)  
  
$todoRepository = $todoRepository;  
$presenterResolver = $presenterResolver;
```

switchがなくなりましたね！
きれいです！
ここまで変わるものなんですね！

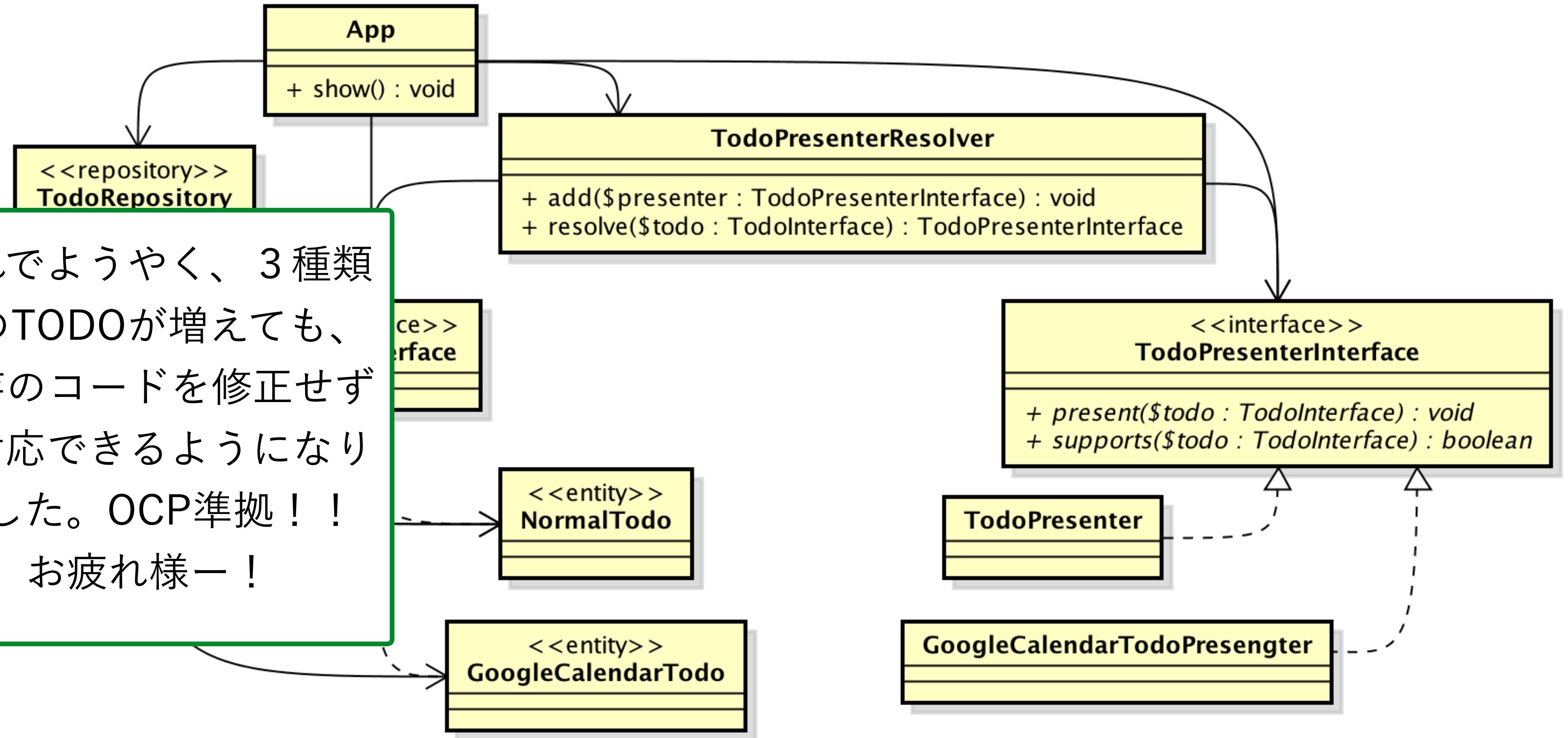
```
public function show()  
{  
    $allTodo = $this->todoRepository->findAll();  
    foreach ($allTodo as $todo) {  
        $presenter = $this->presenterResolver->resolve($todo);  
        $presenter->present($todo);  
    }  
}
```

修正後のクラス図

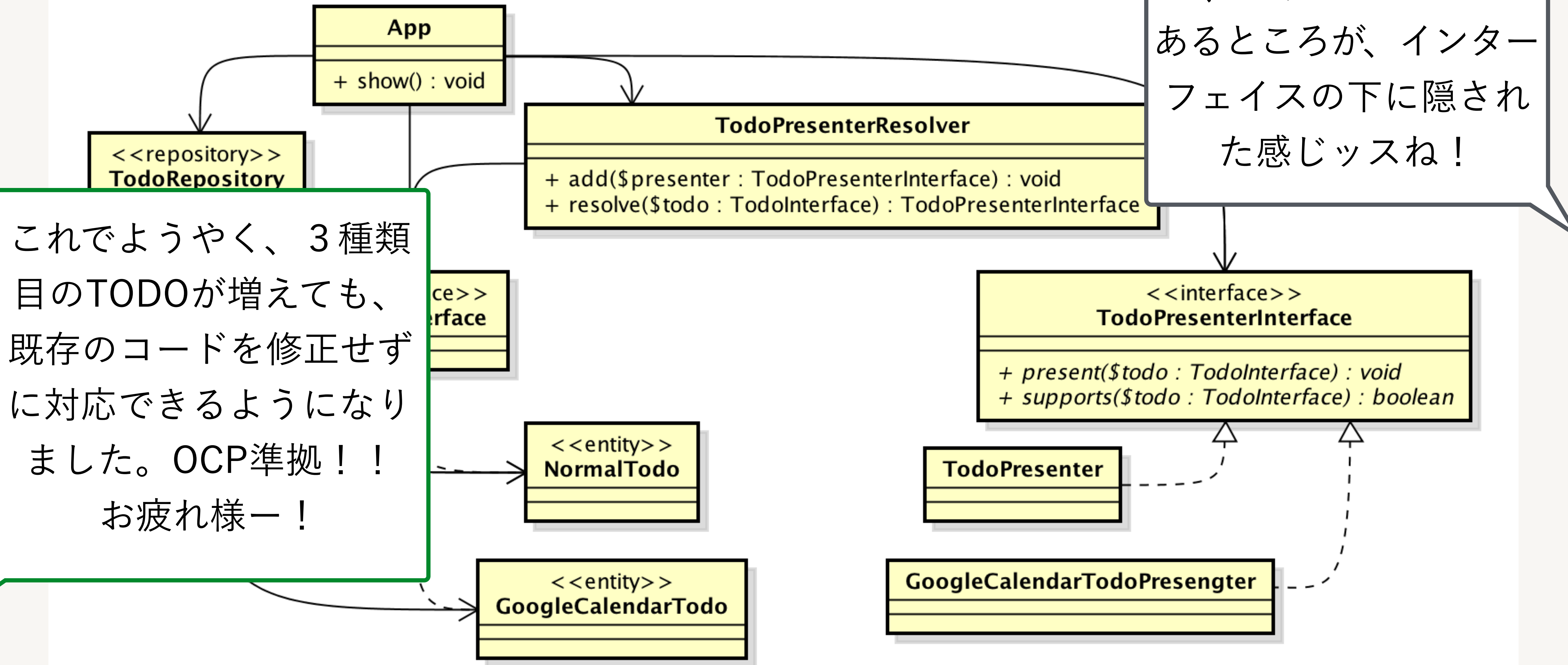


修正後のクラス図

これでようやく、3種類
目のTODOが増えても、
既存のコードを修正せず
に対応できるようになり
ました。OCP準拠！！
お疲れ様ー！



修正後のクラス図



先輩がほとんど書き直してくれた！

めでたしめでたし

・・・😄

新人PHPerの感想

新人PHPerの感想

設計原則って、教条的に従うようなイメージでしか知らなかったなあ。今回の先輩の説明は、そのイメージと全く違った。上手く言葉にできないけど、先輩の中では、コードの問題点と、その改善方法とが、設計原則を通じて有機的に結びついているようで、とても説得力があった。

どうやったらあんな風になれるのか想像できないのだけど、先輩からのアドバイスを頼りに、自分で意識的に原則を使う練習をやってみようと思う。

まとめ3つ

まとめ 3 つ

- オープン・クローズドの原則が、コードに要請すること：

まとめ 3 つ

- オープン・クローズドの原則が、コードに要請すること：
拡張できることと、修正しないこととを、同時に満たせる状態

まとめ3つ

- オープン・クローズドの原則が、コードに要請すること：
拡張できることと、修正しないこととを、同時に満たせる状態
- オープン・クローズドの原則の着眼点：

まとめ3つ

- **オープン・クローズドの原則が、コードに要請すること：**
拡張できることと、修正しないこととを、同時に満たせる状態
- **オープン・クローズドの原則の着眼点：**
バリエーションの軸に沿ったコードのまとめり

まとめ3つ

- オープン・クローズドの原則が、コードに要請すること：
拡張できることと、修正しないこととを、同時に満たせる状態
- オープン・クローズドの原則の着眼点：
バリエーションの軸に沿ったコードのまとめ
- オープン・クローズドの原則に準拠するよう修正するには：

まとめ3つ

- **オープン・クローズドの原則が、コードに要請すること：**
拡張できることと、修正しないこととを、同時に満たせる状態
- **オープン・クローズドの原則の着眼点：**
バリエーションの軸に沿ったコードのまとめり
- **オープン・クローズドの原則に準拠するよう修正するには：**
バリエーションの軸に沿ったコードのまとめ直し

ご清聴

ありがとうございました！

参考文献

- ・ アジャイルソフトウェア開発の奥義 第2版

SOLIDの原則

- 単一責任の原則（SRP：Single Responsibility Principle）
- オープン・クローズドの原則（OCP：Open-Closed Principle）
- リスコフの置換原則（LSP：Liskov Substitution Principle）
- インターフェイス分離の原則（ISP：Interface Segregation Principle）
- 依存関係逆転の原則（DIP：Dependency Inversion Principle）

バリエーションからの保護 (Protected Variation)

- Protected Variation (PV)
- GRASPパターンの1つ
- バリエーション防護パターンという訳
(実践UML)
- 予測されるバリエーション (Predicted Variation) という概念もある
- PV、OCP、Information Hidingは、根本的に同じことを指しているという意見 (同書)