第8章前半
密結合

絡まって解きほぐせない構造

この章の目的

◆密結合が引き起こす問題について考える
◆単一責任の原則について知る

内容

内容とキーワード

◆密結合と責務

◆単一責任の原則

◆DRY原則

結合度と責務とは

結合度と責務とは

◆結合度はモジュール間の、依存の度合いを表す指標。

◆ここではモジュールの粒度をクラスとする。

◆責務とはある関心事について、正常に動作するよう制御する責任のこと。

密結合と責務

密結合と責務

密結合なコードは理解が困難であり、変更が非常にやっかいであるので、結合度の低い疎結合なコードを目指す。

他の多くのクラスに依存してるクラスは修正や機能の追加を行う際に、
意図せぬ影響が生まれることがある。

九龍寨城（通称クーロン城）のようなコードが理解が困難

結合度と責務

◆ECサイトに機能追加する架空のシチュエーションを考えてみる

◆あるECサイトに割引サービスを追加します。通常割引の仕様を以下の様にします。

- ■商品1点につき300円割り引く
- ■上限20,000円まで商品を追加可能

コード

```
class DiscountManager {
     List < Product > discountProducts;
     int totalPrice;
     boolean add( Product product, ProductDiscount productDiscount) {
          if (product. id < 0) { throw new IllegalArgumentException();}
          if (product. name. isEmpty()) { throw new IllegalArgumentException();}
          if (product. price < 0) { throw new IllegalArgumentException(); }
          if (product. id != productDiscount. id) {
               throw new IllegalArgumentException();
          }
          int discountPrice = getDiscountPrice(product.price);
          int tmp;
          if (productDiscount. canDiscount) {
               tmp = totalPrice + discountPrice;
          } else {
               tmp = totalPrice + product. price;
          }
          if (tmp <= 20000) {
               totalPrice = tmp;
               discountProducts.add(product);
               return true;
          } else { return false; }
     }
     static int getDiscountPrice( int price) {
          int discountPrice = price - 300;
          if (discountPrice < 0) {
               discountPrice = 0;
          }
          return discountPrice;
     }
}
```

```
class Product {
     int id; // 商品 ID
     String name; // 商品名
     int price; // 価格
}

class ProductDiscount {
     int id; // 商品 ID
     boolean canDiscount; // 割引 可能な場合 true
}
```

仙塲 大也. 良いコード／悪いコードで学ぶ設計入門–保守しやすい 成長し続けるコードの書き方 (pp.245-246). 株式会社技術評論社. Kindle 版.

結合度と責務

◆商品割引に関するクラス

DiscountManagerクラス

- 商品リスト
- 総額
- 商品を追加するメソッド
- 割引価格を取得するメソッド

Productクラス

- 商品ID
- 商品名
- 価格

○商品を追加するメソッドの仕様
・Productの不正チェック
・割引価格の計算
・割引可能であれば割引価格を総額に加算
　そうでなければ通常価格を総額に加算
・総額20,000円以内であればリストに追加

ProductDiscountクラス

- 商品ID
- 割引可能か（Boolean）

結合度と責務

◆さらに、通常割引の他に以下の夏期限定割引の仕様を追加しようとします

- ■商品1点につき300円割り引く（通常割引と同じ）
- ■上限30,000円まで商品を追加可能（1万円多い）

◆これをSummerDiscountManagerクラスを作成して実装することにします

結合度と責務

◆夏季限定割引に関するクラス

DiscountManagerクラス

DiscountManagerオブジェクト

DiscountManagerの
割引計算メソッドを流用する

商品を追加するメソッド

Productクラス

商品ID

商品名

価格

夏季割引可能か(Boolean)

新しく変数を追加

○商品を追加するメソッドの仕様
・Productの不正チェック（再定義）
・割引価格の計算（流用する）
・割引可能であれば割引価格を総額に加算
　そうでなければ通常価格を総額に加算
・総額30,000円以内であればリストに追加

コード

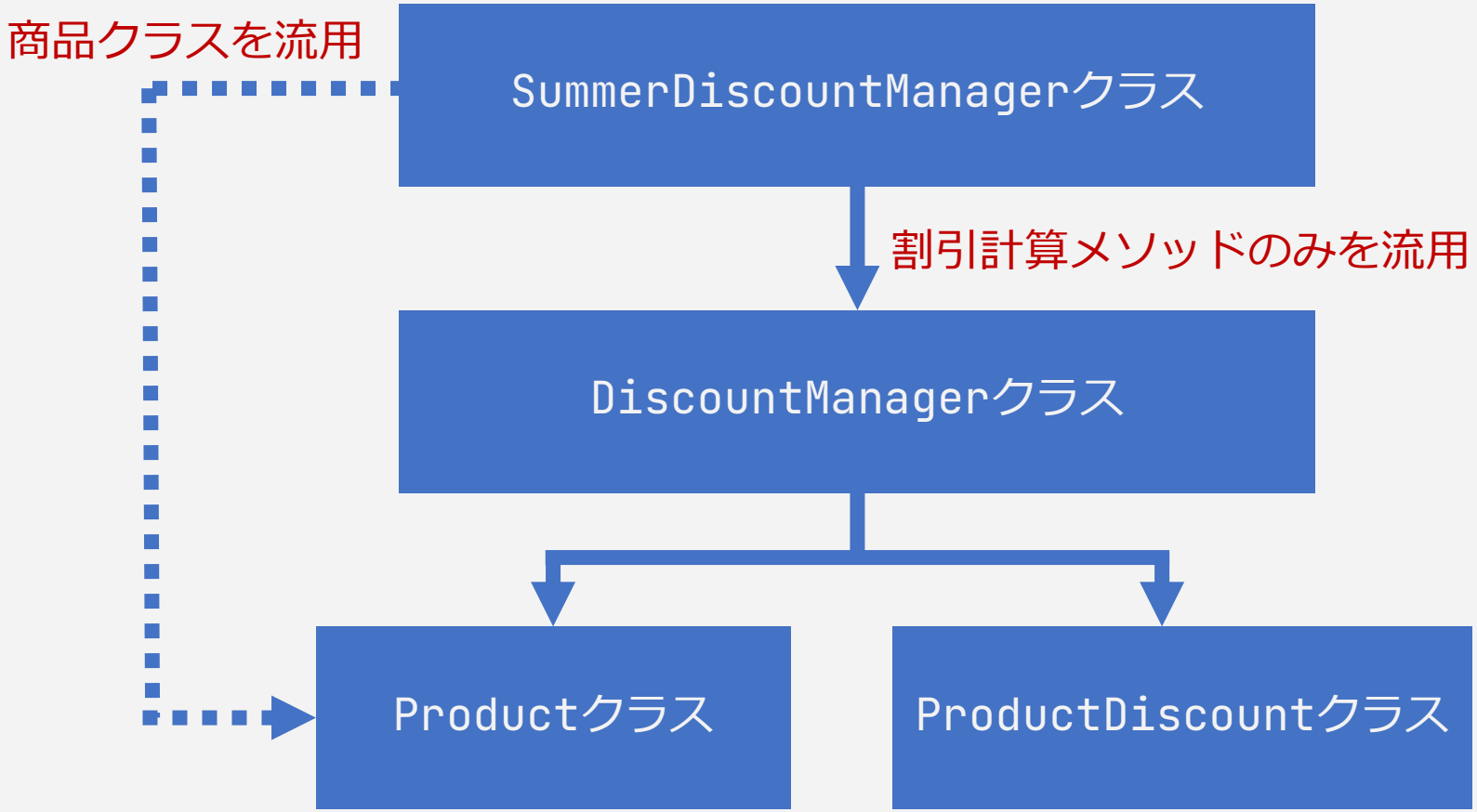
```
class SummerDiscountManager {
     DiscountManager discountManager;
     boolean add( Product product) {
          if (product.id < 0) {
               throw new IllegalArgumentException();
          }
          if (product.name.isEmpty()) {
               throw new IllegalArgumentException();
          }
          int tmp;
          if (product. canDiscount) {
               tmp = discountManager.totalPrice
                         + DiscountManager.getDiscountPrice(product.price);
          } else {
               tmp = discountManager. totalPrice + product. price;
          }
          if (tmp < 30000) {
               discountManager.totalPrice = tmp;
               discountManager.discountProducts.add(product);
               return true;
          } else {
               return false;
          }
     }
}
```

```
class Product {
     int id; // 商品 ID
     String name; // 商品名
     int price; // 価格
     boolean canDiscount; // ←夏季割引可能か
}
```

仙塲 大也. 良いコード／悪いコードで学ぶ設計入門–保守しやすい 成長し続けるコードの書き方 (p.248). 株式会社技術評論社. Kindle 版.

結合度と責務

◆割引に関するクラスの関係図



結合度と責務

◆ここで仕様変更の必要が出てきて…


```
static int getDiscountPrice(int price) {
    int discountPrice = price - 400;
    if (discountPrice < 0) {
        discountPrice = 0;
    }
    return 0;
}
```

通常割引価格を300円から
400円に変更するぞ！

これで通常割引価格を400円に
変更できましたが…

結合度と責務

◆仕様変更後システム利用者から苦情が…

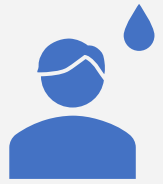
通常割引価格だけじゃ無くて
夏季限定割引価格まで400円に
変更されているじゃないか！

マイナス価格の商品を夏季割引に
追加できるようになっているぞ！

割引対象商品に設定したのに
割引されていないぞ！
どうなってる！

利用者

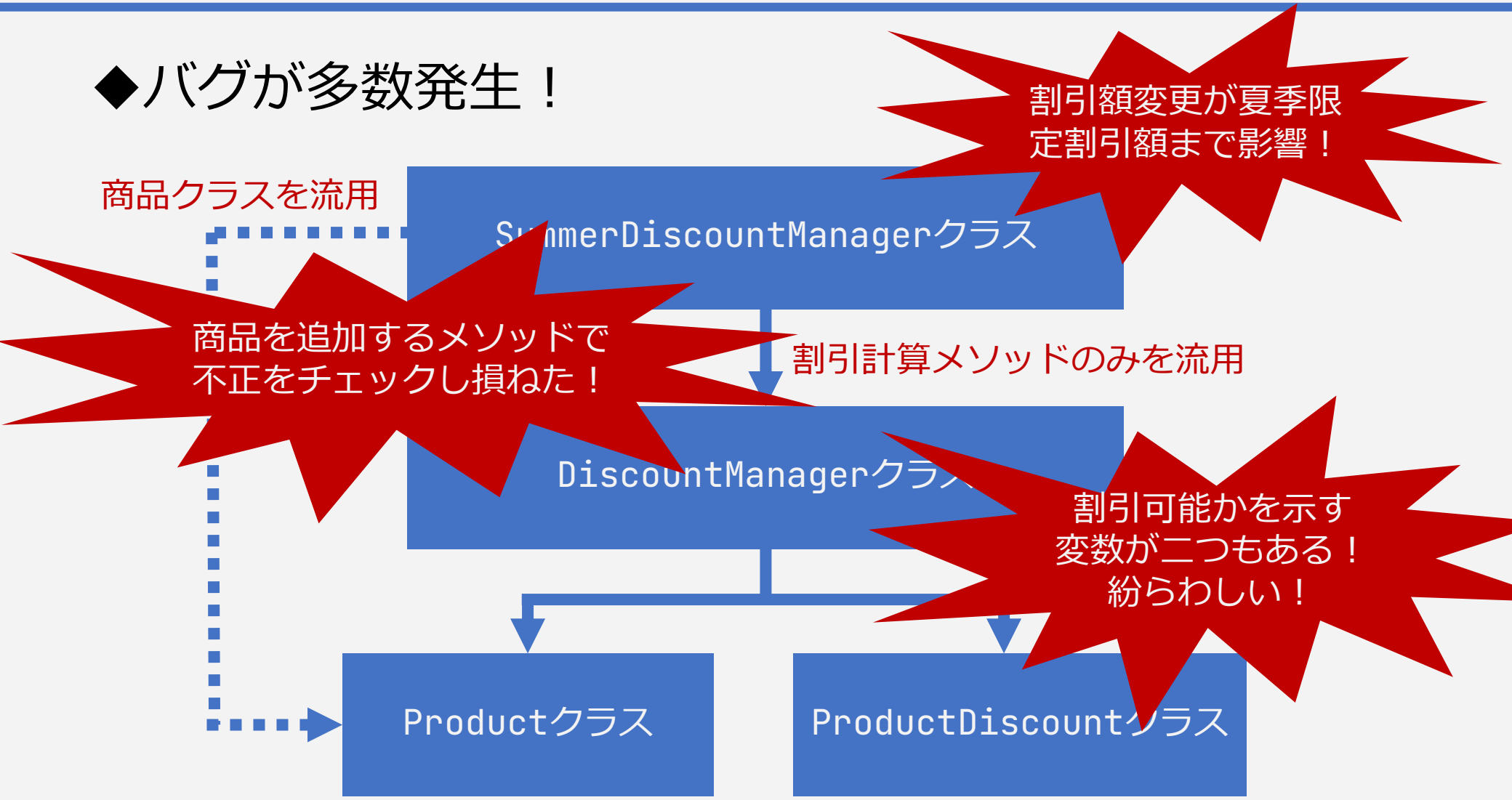
え？？確認します…



開発者

結合度と責務

◆バグが多数発生！



結合度と責務

先ほどのクラスは責務が考慮されていない。ロジックの置き場所がちぐはぐである。

◆先ほどのクラスの問題点

■DiscountManagerクラスで多くの処理を任せている

- 責務が多すぎる

■Productクラスが持つべきメソッドが散在している

- 低凝集

■割引可能を示す変数の名前が酷似していた

- 別のクラスで同じ名前の変数を扱っていた

■夏季限定割引価格の計算のために通常割引価格の計算ロジックを安易に流用した

- 都合良く無理矢理使用している

単一責任の原則

単一責任の原則

厳密には違いますが、ここでは責任=責務とします。

◆単一責任の原則とは

■クラスが担う責任は、たったひとつに限定すべき

◆先ほどのクラスは…

■DiscountManagerの割引価格計算メソッドは通常割引価格の計算だけでは無く、夏季限定割引価格の計算を行う責務を持っており、2重に責務を背負わされている

■DiscountManagerクラスはProductの不正をチェックしており、Productクラスが背負うべき責務を代わりに負っている。

⇒ このように、DiscountManagerは単一責任の原則に違反している

単一責任の原則

◆責務が単一になるようにクラスを設計しよう

◆先ほどのクラスを書き換えてみましょう…

■まず定価クラスを用意します

- 定価について責務（不正状態のチェック等）を持たせる

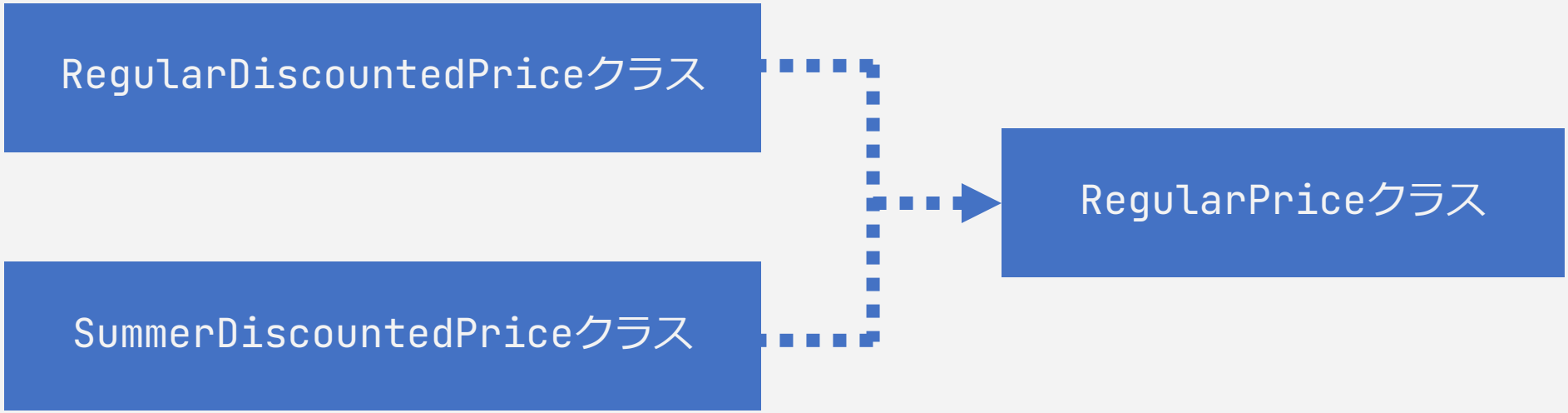
■通常割引価格と夏季限定割引価格のクラスも作成

- 異なる概念ごとに丁寧にクラス化する

単一責任の原則

結局どうなったか。

クラスが通常割引価格と夏季限定割引価格の責務毎に別れている



⇒ それぞれの仕様が変更されても互いに影響はない！
⇒ このような構造を疎結合と呼ぶ

DRY原則の誤用

DRY原則の誤用

◆DRY原則とは

■Don't Repeat Yourselfの略で「繰り返しを避けよ」と訳される。

■一部では「コードの重複を許すな」といった解釈が広まっているが、そうではない。

■「新装版　達人プログラマー　職人から名匠への道」では以下の様に説明されている

「全ての知識はシステム内において、単一、かつ明確な、そして信頼できる表現になっていなければならない」

知識とはソフトウェアで扱うビジネス概念などがある

DRY原則の誤用

- ◆同じようなロジック、似ているロジックであっても、概念が違えばDRYにすべきではない
- ◆先ほどのクラスでは…
  - ■通常割引価格と夏季限定割引価格はそれぞれ別の概念です
  - ■概念が異なるもの同士を無理にDRYしようとすると密結合になってしまう

次の章に向けて

◆密結合について学んだ

◆密結合の各種事例の対処法について次章で学ぶ