プログラム設計

http://bit.ly/design4d

デザインパターン

後期 第13週

2019/12/23

デザインパターンとは

ソフトウェアの設計
(特ににクラスの概念を用いた設計)

↓

どのような**目的**の時に、
どのような**役割**のクラスを定義するとよいのか

↓

生産性の高いソフトウェア開発

デザインパターンとは

どのような**目的**の時に、
どのような**役割**のクラスを定義するとよいのか

「MVC」という考え方もこの一例

他の目的に適した良い設計方法は？

自分で試行錯誤して考えると
手間も時間もかかる・・・

デザインパターンとは

デザインパターン

- よく使われる「目的」ごとにパターンを分類
- パターンごとに設計方法がまとめられている
- 自分のソフトウェア開発効率が上がる

デザインパターンの参考書

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides 著

  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

  ソフトバンクパブリッシング

  （デザインパターンの考案者達による解説本。23種類のパターンが解説されている。）

- 結城 浩 著

  Java 言語で学ぶデザインパターン入門

  ソフトバンクパブリッシング

  （上記のパターンを Java言語によるコードを用いて解説している。）

- 高橋 麻奈 著

  やさしいJavaオブジェクト指向編

  ソフトバンククリエイティブ

  （具体的な例の中に色々なパターンを混ぜて説明している。
  後述の例もこれを参考にしている。）

今回は

デザインパターンの例として、

「Observerパターン」を学びます

MVCでは・・・

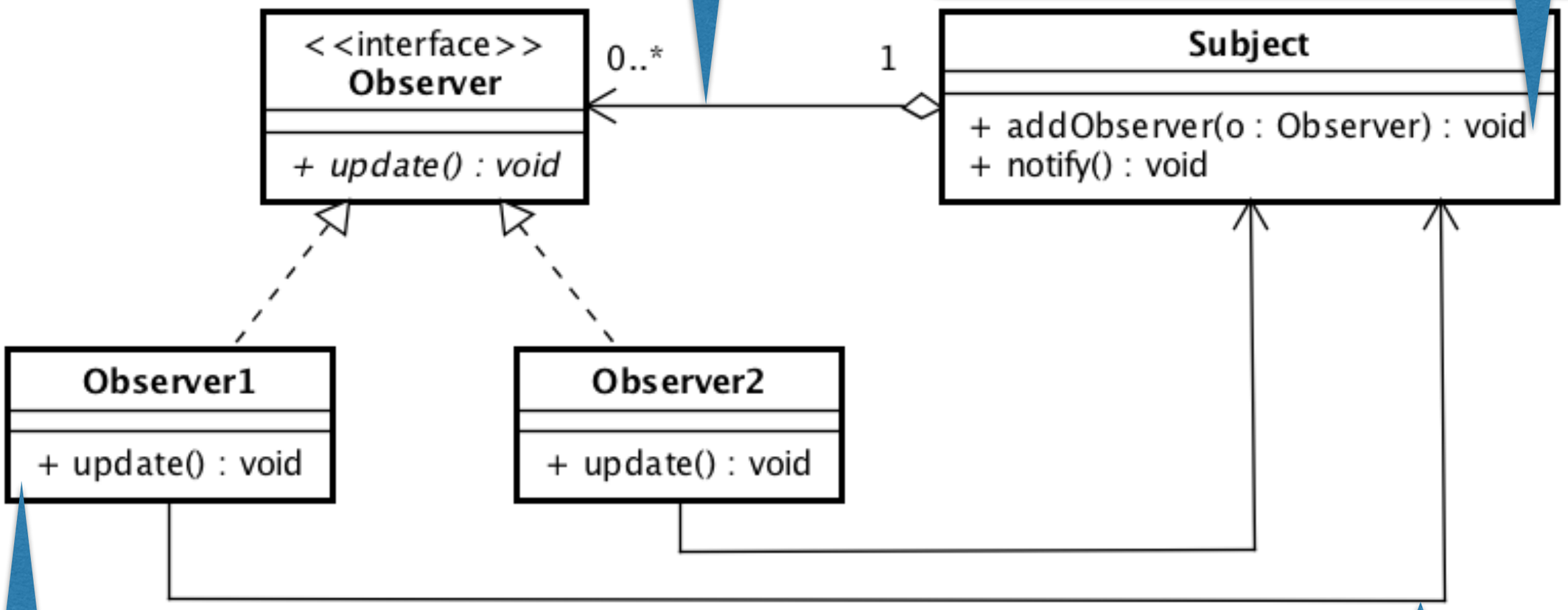
「ModelとViewの関係において、Model が変更された際の

Viewへの通知」の部分に活用することができる

Observerパターン

Subjectが**複数個**のObserverを集約できる

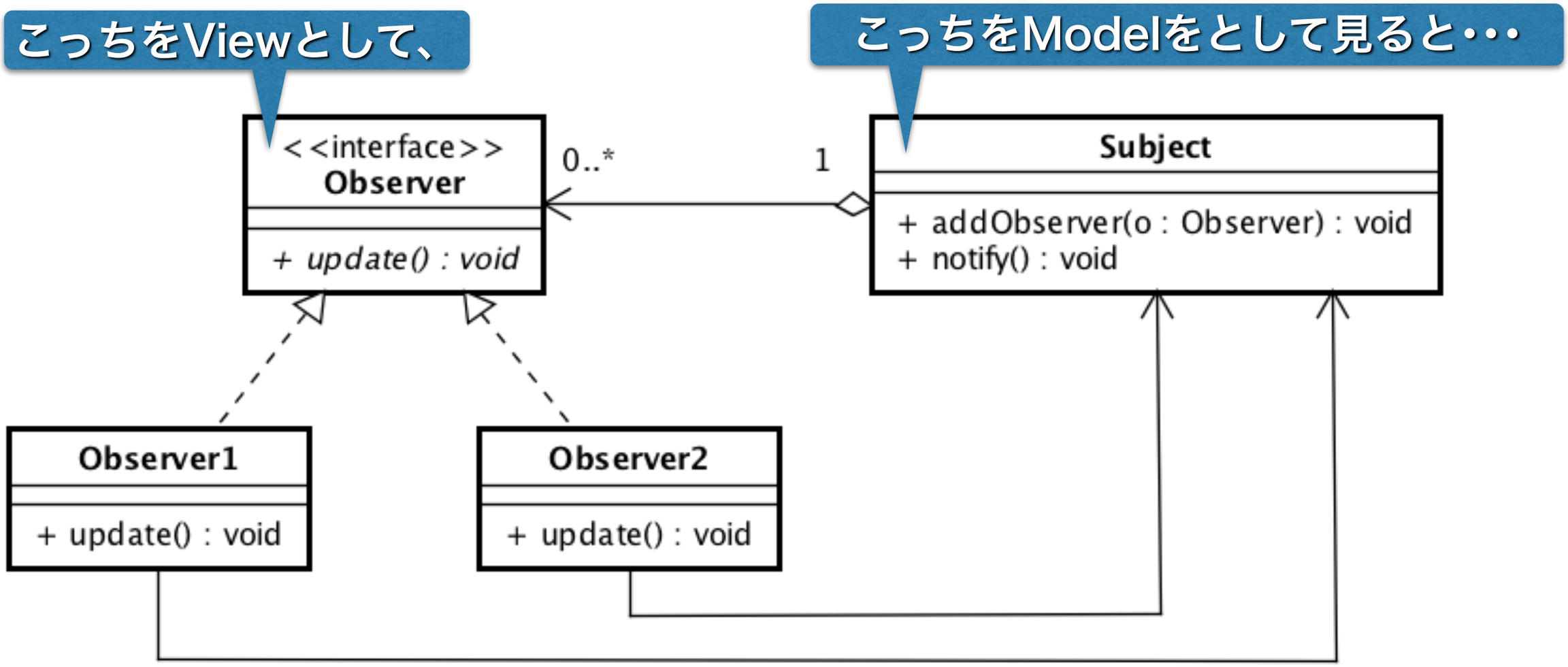
addObserverで複数のObserverを登録できる



Subjectは、**インタフェースObserverを実装（implements）したクラス**（ここではObserver1, Observer2）なら、いくつでも集約可能

Subjectへ**通知を送る（notifyを呼び出す）**をために、Subjectを参照している

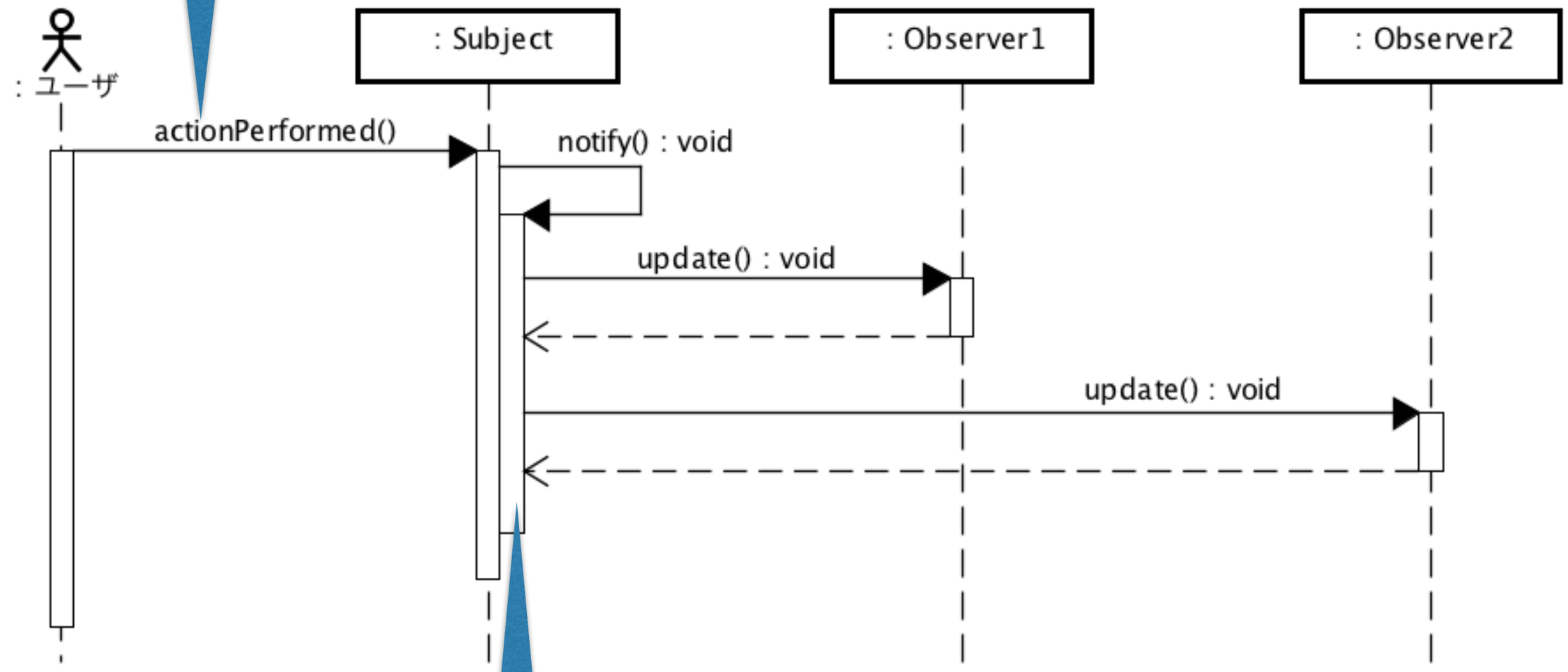
Observerパターン



「1つのModelに対して、複数のViewを持たせる」
という設計が楽にできるようになる

Observerパターン

外部から何か操作があると呼び出される



通知を受け取ったら、集約している全てのObserverに更新（update）を送る

【課題の準備】

演習室で作業する前に、以下のコマンドを
入れるだけで準備が完了する

```
$   mygitclone4d   「自分のGitHubユーザ名」
$   cd prog4d-(ユーザ名)
$   ./myconf
```

※本体をシャットダウンするまでは、
上記「mygitclone」と「myconf」の設定は有効です

【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して後期第13週のフォルダを作る

```
$ cd prog4d-(ユーザ名)        (←既に移動しているなら不要)
$ mkdir week213
$ cd week213
```

※講義資料で同じ場所で配布されている

「week213.zip」をダウンロードして解凍して下さい。

【課題13-1】

「データをグラフ描画する」アプリケーションを実行してみます。準備でダウンロードした フォルダに含まれている以下のファイルの内容を確認して下さい。

- SubjectData.java（Subjectクラス）
- Observer.java（Observerインタフェース）
- ObserverList.java（リスト表示するObserverクラス）
- ObserverBarGraph.java（棒グラフ表示するObserverクラス）
- ObserverLineGraph.java（折線グラフ表示するObserverクラス）
- ObserverMain.java（main を持ったクラス）
- data1.txt（読み込むサンプルデータ その1）
- data2.txt（読み込むサンプルデータ その2）

※Observerパターンに関係している箇所は、
コメント中に「今日のPoint」として説明を入れてあります。

【課題13-1】

全てのJavaファイルをコンパイルして、アプリケーションを実行し、サンプルデータを読み込ませて結果を確認しましょう。（このアプリケーションでは、リスト表示と棒グラフ表示がObserverとしてSubjectに登録されています。）

【課題13-2】

折線グラフを表示するクラスObserverLineGraphを、クラスSubjectDataのObserverに登録して、このアプリケーションに折線グラフ表示機能も追加してみましょう。

クラスObserverMainのmainメソッドに、以下のような処理を追加します。（既に追加されているObserverListとObserverBarGraphを参考に追加できます。）

- ObserverLineGraphのインスタンスを1つ作る。
- このインスタンスのsetSubjectDataメソッドを使って、Subjectインスタンスを参照する。
- addObserverメソッドを使って、このインスタンスをObserverとして追加する。

【課題13-2】

コンパイルと実行について

SubjectMainクラスのみをコンパイルするだけで、他のJavaファイルは改めて変更・コンパイルする必要はありません。このアプリケーションが完成すると、リスト表示と棒グラフ表示に加えて、折線グラフ表示が追加されます。サンプルデータを読み込ませて結果を確認しましょう。

【課題の提出】

以下の流れで、GitHubにプッシュしてWebサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m “課題13-2提出”
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

https://github.com/nit-ibaraki-prog4d-2019/prog4d-(ユーザ名)

【課題13-3】

課題13-2で作ったアプリケーションを更に改良して、棒グラフの表示と折線グラフの表示が2個ずつになるように、表示するフレームを増やしてみて下さい。

- ObserverBarGraphとObserverLineGraphのインスタンスをそれぞれもう一つ増やす。
- 前問と同様に、参照関係とObserverの追加をする。

同じクラスから作成されたインスタンスをObserverとして追加しても、それぞれが個々のViewとして動作することが確認できます。

【課題の提出】

以下の流れで、GitHubにプッシュしてWebサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m “課題13-3提出”
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

https://github.com/nit-ibaraki-prog4d-2019/prog4d-(ユーザ名)

【課題13-4】

課題13-3で作ったアプリケーションを更に改良して、ObserverBarGraph, ObserverLineGraphとは異なる色でグラフを描画するクラスを増やし てみて下さい。

- ObserverBarGraph, ObserverLineGraphをコピーして、新しいクラス名に変える。
- paintメソッド内にある、色に関する箇所を別の色に変更する。
- mainメソッドに、前問と同様、参照関係とObserverの追加をする。

Observerを増やす場合は、新しく作ったObserverのクラスのメソッドupdate（または、そこから呼び出されるpaintのような描画メソッド）の処理をそれに合わせて作り直せば良いことがわかります。

【課題の提出】

以下の流れで、GitHubにプッシュしてWebサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m “課題13-4提出”
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

https://github.com/nit-ibaraki-prog4d-2019/prog4d-(ユーザ名)