Design Patterns - TP5

https://github.com/IUT-Blagnac/cpoa-tp4-template

Code initial pour le TP5

Rappel du cours : http://bit.ly/jmb-cpoa

Informations générales

NOM

BRUEL

Prénom

Jean-Michel

Groupe #

[X] Enseignants

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] Innopolis

Pré-requis

Il vous faut :

- [x] Un compte GitHub
- [ ] Un terminal de type Git Bash (si vous utilisez Window$)

Essayez la commande suivante dans votre terminal pour vérifier votre environnement `git` :

```
git config --global -l
```

Tâche initiale

- [x] Clickez sur le lien Github Classroom fourni par votre enseignant (en fait c'est déjà fait si vous lisez ces lignes).

- ☐ Clonez sur votre machine le projet Github généré pour vous par Github Classroom.
- ☐ Modifez le `README` pour modifier Nom, Prénom et Groupe.
- ☐ Commit & push:

commit/push

```
fix #0 Initial task done
```

Dans la suite de ce document, à chaque fois que vous trouverez un énoncé commençant par `fix #⋯` vous devez vérifier que vos scripts/fichiers modifiés sont bien dans votre dépôt local en vue de committer et de pusher les modifications sur votre dépôt distant en utilisant comme message de commit cet énoncé.

- Si vous voulez vérifier que vous êtes prêt pour le `fix #0`, utilisez la commande : `make check`.
- Si vous voulez avoir la liste des ToDos de ce TP/TP, exécutez `make todos`.

Récupérer l'archive de projet Eclipse ObserverInitialCode.zip. L'importer dans un workspace Eclipse.

1. Lire rapidement les et comparer les applications `observer.nonpattern` et `observer.pattern`.
2. Lancer les 2 applications (`CourseViewer.main()` et `CourseController.main()`) et observer ce quelles affichent.

1. Exercices

1.1. Reverse Engineering

1. Obtenez et comparez les 2 diagrammes de classes des deux packages (`TP5.plantuml` et `TP5-observer.plantuml` à placer dans le répertoire `docs`).

Vous devez obtenir les 2 diagrammes suivants :

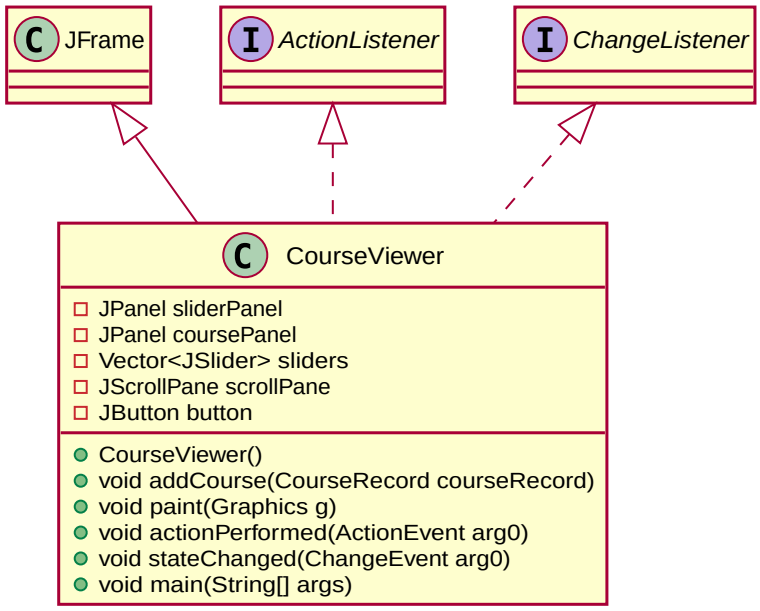

Figure 1. `observer.nonpattern`

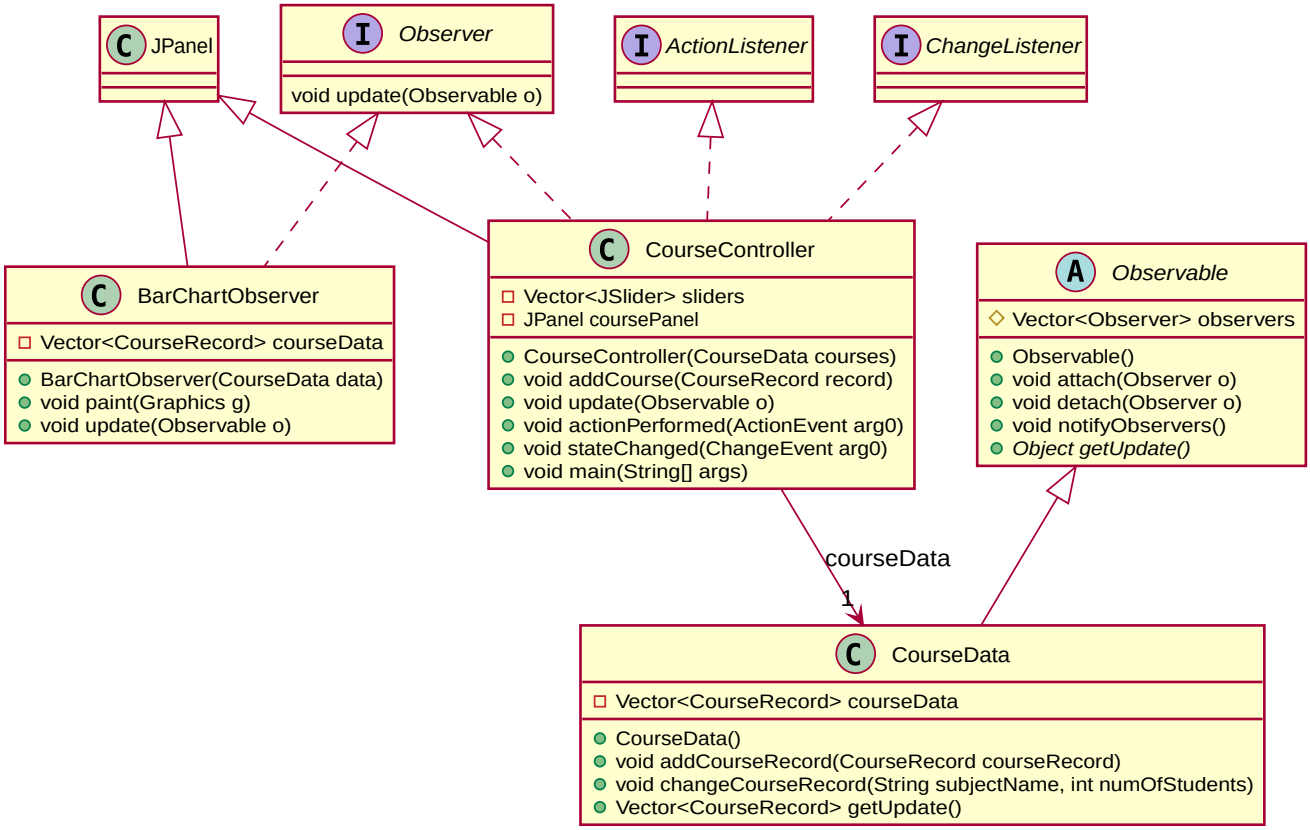

Figure 2. `observer.pattern`

commit/push

```
fix #1.1: Add Class Diagrams
```

1.2. Evolution du code `observer.nonpattern`

Etendre le code `observer.nonpattern` pour présenter le vecteur de `CourseRecords` comme un camembert (*pie chart*), en plus de la forme actuelle (*bar chart*).

Allez à l'essentiel, le but n'est pas d'avoir un super code, mais de vous confronter à la difficulté d'évolution. Pour plus de détails sur les camembers en Swing, cf. section finale.

commit/push

```
fix #1.2: Add Pie Chart in nonPattern
```

1.3. Evolution du code `observer.pattern`

Faites la même chose avec le code `observer.pattern`

Que pensez-vous de la différence (en terme d'effort de codage et de résultat)?

commit/push

```
fix #1.3: Add Pie Chart in Pattern
```

1.4. Push vs. Pull

Vous avez pu observer que la version du patron *Observer* utilisait un modèle *pull* (`notifyObservers()` n'envoie aucune information).

Réalisez une version `push`.

commit/push

```
fix #1.4: Add push version
```

1.5. Inconvénient du *push*

Dans cette nouvelle version, si les programmes ont plus de 1000 cours, et si l'un seulement évolue, `notifyObservers()` pousse toutes les informations sur tous les observateurs!

Améliorez votre modèle *push* pour qu'il ne pousse que les données pertinentes.

Pour cet exercice, vous pourrez ignorer les changement de `New Course` et continuer à utiliser le modèle *pull* pour ce type de changement.

commit/push

```
fix #1.5: Change notify for smart push
```

1.6. Sélection des *updates*

Vous aurez remarqué que `CourseController` n'est intéressé que par les changements de `New Course`, alors que `BarChart` et `PieChart` ont besoin de connaitre les changements de valeurs de `JSlider`. Etendre l'interface d'inscription de `Observable` (la méthode `attach`) pour que `CourseController` ne reçoivent plus les mises à jour des `updates` qui ne l'intéressent pas.

commit/push

```
fix #1.6: Add Smart attach()
```

How to draw a pie chart ?

Here is a code segment that draws a pie chart given a Graphics object and an Array containing Integers to be represented in the pie chart. It is drawn at location (xOffset, yOffset) and with the radius specified to be of size 100.

```
public void paint(Graphics g, Integer[] data) {
    super.paint(g);
    int radius = 100;

    //first compute the total number of students
    double total = 0.0;
    for (int i = 0; i < data.length; i++) {
        total += data[i];
    }
    //if total == 0 nothing to draw
    if (total != 0) {
        double startAngle = 0.0;
        for (int i = 0; i < data.length; i++) {
            double ratio = (data[i] / total) * 360.0;
            //draw the arc
            g.setColor(LayoutConstants.subjectColors[i%LayoutConstants.subjectColors
.length]);
            g.fillArc(LayoutConstants.xOffset, LayoutConstants.yOffset + 300, 2 *
radius, 2 * radius, (int) startAngle, (int) ratio);
            startAngle += ratio;
        }
    }
}
```

commit/push

```
fix #All: Completed all duties
```

Contributeurs

- Jean-Michel Bruel

À propos...

Baked with Asciidoctor (version 2.0.11) from 'Dan Allen', based on AsciiDoc. 'Licence Creative Commons'.  licence Creative Commons Paternité - Partage à l'Identique 3.0 non transposé.