Travail Écrit n°2

Aides autorisées : Toute documentation imprimée, électronique ou accessible par Internet

La communication avec d'autres personnes est interdite (mail, sms, chat, etc). Toute tricherie sera sanctionnée par la note 1.

Durée : 1h. Écrire votre programme dans un seul fichier et l'envoyer à marcel.graf@heig-vd.ch

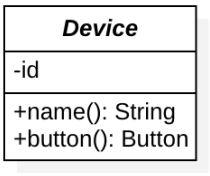
Domotique

Un système de domotique définit l'interface `Button` ci-après pour représenter tous les boutons qui ont deux surfaces arrangées de façon verticale. On peut tapoter *(to tap)* les surfaces.

```
interface Button {
    void tapUpper();
    void tapLower();
}
```

Le système est composé de dispositifs (classe `Device`, dont l'UML incomplet et montré ci-après).

- On peut commander les dispositifs avec des boutons que l'on peut obtenir avec la méthode `button()`.
- À chaque dispositif est attribué un identificateur unique.



1. Lumières

Une lumière (classe `Light`) est un dispositif.

- Une lumière peut être éteinte *(Off)* ou allumée *(On)*. Quand elle est créée, elle est éteinte.
- Un tapotement sur la surface supérieure du bouton allume la lumière, sur l'inférieure l'éteint. Quand elle change d'état, la lumière émet un message correspondant. Si elle se trouve déjà dans l'état désiré, elle émet un message que le tapotement a été ignoré.

Définir la classe `Light` et la classe `LightState` pour représenter son état, et compléter la classe `Device` si nécessaire, afin que le code ci -après

```
System.out.println("### Lights");
Device d1 = new Light();
System.out.println(d1);
d1.button().tapUpper();
d1.button().tapUpper();
Device d2 = new Light();
System.out.println(d2);
```

produise l'affichage

```
### Lights
Light #1 is Off
OK, Light #1 is On
Ignored, Light #1 is On
Light #2 is Off
```

2. Stores

Un store (classe `Blind`) est un dispositif.

- Un store peut être fermé *(Closed)*, semi-ouvert *(HalfOpen)* ou ouvert *(Open)*. Quand il est créé, il est ouvert.

- Un tapotement sur la surface supérieure du bouton fait que le store s'ouvre davantage (s'il est fermé, il devient semi-ouvert ; s'il est semi-ouvert, il devient ouvert). La surface inférieure à l'effet inverse. Quand le store change d'état, il émet un message correspondant. S'il est déjà fermé et que l'on tapote la surface inférieure, il émet un message que le tapotement a été ignoré. Comportement correspondant pour l'autre direction.

Définir la classe `Blind` et la classe `BlindState` pour représenter son état, et compléter la classe `Device` si nécessaire, afin que le code ci-après

```
System.out.println("\n### Blinds");
Device d3 = new Blind();
System.out.println(d3);
Device d4 = new Blind();
System.out.println(d4);
d4.button().tapLower();
d4.button().tapLower();
d4.button().tapLower();
```

produise l'affichage

```
### Blinds
Blind #3 is in position Open
Blind #4 is in position Open
OK, Blind #4 is in position HalfOpen
OK, Blind #4 is in position Closed
Ignored, Blind #4 is in position Closed
```

3. État du système

Définir la méthode `showSystemState()` de la classe `Device` qui affiche tous les dispositifs et leurs états. Un appel de la méthode produit l'affichage ci-après.

```
State of system:
- Light #1 is On
- Light #2 is Off
- Blind #3 is in position Open
- Blind #4 is in position Closed
```

4. Égalité

Deux dispositifs sont considérés égaux s'ils sont du même type et dans le même état. Définir des méthodes dans les classes `Light` et `Blind` qui permettent de le vérifier. Définir une méthode qui permet de chercher parmi tous les dispositifs ceux qui sont égaux à un dispositif donné.

Le code

```
System.out.println("\n### Find device");
Device.findDevice(new Light());
Device.findDevice(new Blind());
```

produira l'affichage

```
### Find device
Devices equal to Light #5 is Off:
- Light #2 is Off
Devices equal to Blind #6 is in position Open:
- Blind #3 is in position Open
```