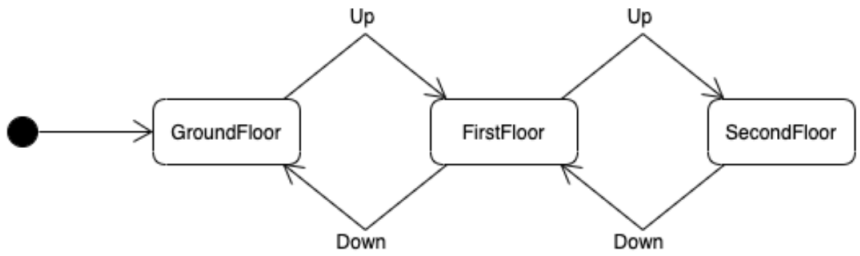
Design Patterns em Java

Introdução às FSM (Finite-State Machine)

Exercício

- Resolva o exercício 24 da ficha de exercícios

24. Considere o seguinte diagrama de estados, que descreve o funcionamento de um elevador num prédio com 3 pisos.



a. Construa as classes necessárias para implementar esta máquina de estados.
   i. Em cada classe que representa um estado, deve criar métodos adequados para representar as transições possíveis a partir do respetivo estado (*Up* e/ou *Down*).
   ii. Desenvolva uma classe que represente a interação com o utilizador, onde é permitido desencadear as ações (*Up* e *Down*) sobre a máquina de estados, independentemente do seu estado atual (justificação: para verificação da robustez da máquina de estados, perante ações não previstas)
   iii. Altere a interação com o utilizador, criando métodos que apresentem as opções adequadas para cada estado (i.e: possibilidade de carregar nos botões para subir ou descer no 1º piso, mas apenas carregar no botão para subir no R/C).

Resolução

- Criar classe `Elevator` que permita representar os dados do elevador
  - Na primeira versão a única informação a gerir é o piso em que o elevador se encontra
- Criar `enum ElevatorState` com o estados possíveis
- Criar `interface IElevatorState` com os métodos correspondentes às transições entre estados
  - Incluir também método `getState` que permita retornar o estado atual (constante definida pelo *enum*)

Resolução

- Criar classe `ElevatorContext` a qual deve incluir:
  - Referência para o estado atual
    - Criar no construtor
  - Referência para o modelo de dados
    - Criar no construtor
  - Método público que permita obter o estado atual
  - Método *package-private* que permita alterar o estado atual
  - Métodos que reencaminhem as ações/eventos para o estado ativo
  - Conjunto de métodos que permitam obter os dados necessários à interação com o utilizador ou com os restantes módulos do programa
- A classe `ElevatorContext` deve garantir que as alterações ao modelo de dados apenas ocorrem no contexto dos métodos referentes a transições de estado
  - Não deve disponibilizar métodos ou retornar referências para objetos que permitam alteração direta dos dados

Resolução

- Criar classe abstrata `ElevatorStateAdapter` que implemente a interface anterior e que permita
  - Fornecer implementações por omissão para todas as transições
  - Gerir referências para a classe base que representa o modelo de dados e para a classe que representa o contexto geral da máquina de estados
  - Fornecer método para alterar o estado atual no contexto

- Criar classes derivadas da classe `ElevatorStateAdapter`, que representem cada um dos estados concretos
  - `GroundFloorState`
  - `FirstFloorState`
  - `SecondFloorState`

Resolução

- Main

```
public class Main {
    public static void main(String[] args) {
        ElevatorContext fsm = new ElevatorContext();
        ElevatorUI ui = new ElevatorUI(fsm);
        ui.start();
    }
}
```

Resolução

- `ElevatorUI`
  - Classe que deverá disponibilizar uma interface adequada para os estados definidos
  - Esta classe não deve implementar qualquer lógica referente às regras da aplicação
  - Deve...
    - apresentar a informação útil para o utilizador efetuar as suas opções
    - fornecer os mecanismos de interação que permitam despoletar as transições de estado adequadas