CES28: MVP exemplo Automato

Atividade em sala, entregar via github

1) ler o codigo e verificar:

a) as camadas estão bem separadas.

V depende de P, P depende de M
Mas: P nao depende de V, M nao depende de P.
Alem disso, V nao depende de M
Não depender significa que nenhuma classe é importada.

O conceito de camadas implica em callbacks. P deve chamar metodos da View.
Portanto, P inclui uma interface abstrata para a View.
Para implementar uma View concreta, precisamos implementar esta interface abstrata.

Além disso, este exemplo mostra que a interação entre camadas pode ser mais complexa do que simplesmente passar um inteiro.

b) apenas a View depende do swing.

A camada View depende da lib de GUI do JAVA, mas não o Presenter ou Model. Todo o código dependente de GUI ficou separado.
Se quisermos fazer outra view com outra lib de janelas, ou sem janelas, nada muda em P ou M. Por exemplo, uma outra View que apenas imprime no console, ou uma View que envia os dados para um servidor remoto.

2) Modificar para usar Observable/Observer do Java

Existe uma interface abstrata entre V e P, e o Presenter possui um ponteiro para uma View, que utiliza para atualizar a View. Mas assim, não podemos usar várias Views ao mesmo tempo com o mesmo Presenter. Usar as classes Java Observable/Observer de forma a poder utilizar varias views ao mesmo tempo

Presenter será o Observable, a View o Observer.
De novo, este exemplo mostra que a interação entre camadas pode ser mais complexa do que simplesmente passar um inteiro. Há duas opções:
Continuar usando as mesmas interfaces em Presenter e View. Presenter envia um ponteiro para si mesmo quando notifica os seus observers. Cada observer usa este ponteiro para se

enviar ao Presenter que o atualiza. A responsabilidade de atualizar a view fica no Presenter, mas isso não é errado pois o Presenter já conhece a interface abstrata da View.
Ou criar métodos no Presenter para permitir que as Views recebam as informações necessárias. Assim a responsabilidade de atualizar a View fica na própria View.
NOTE: a View não pode enxergar a classe Cell, pois Cell pertence ao Model!!!
Uma View que imprime uma matriz de inteiros é mais generica e reusavel do que uma view que imprime celulas. As vezes é tentador fazer todos conhecerem todos os detalhes do problema, mas o conceito de camadas deve isolar.
Pense no exemplo de uma GUI que desenha um semaforo com luzes verde, amarela, vermelha. Se associarmos a interface que comanda esta GUI com metodos como pare(), atencao(), avance(), fica dificil reusar em outro problema onde as cores significam outra coisa, por exemplo ‘ok’, ‘quase no limite’, ‘passou do limite’. O Presenter existe justamente para fazer o meio de campo entre Modelo e View.

3) implementar uma View verdadeira, funcional, mas sem janelas.

apenas imprima a matriz de celulas no console, e peça comandos no teclado para o comando NEXT.
Deve importar a camada Presenter e implementar a mesma interface abstrata IBoardView.

4) realizar testes com uma view mockada.

Agora crie um mock para a View, e teste.