Sobre os padrões de projetos e conceitos de OO vistos em sala de aula, responda cada uma das questões a seguir.

a) Explique com suas próprias palavras como o padrão **STATE** funciona. Utilize exemplos.

b) Explique com suas próprias palavras como o padrão **Mediator** funciona. Como ele pode ser aplicado para resolver problemas de alto acoplamento entre classes em um sistema OO? Explique.

c) Explique com suas próprias palavras a diferença entre **acoplamento** e **coesão** em classes. Qual seria a relação entre estes **conceitos** e os **padrões de projetos**vistos em sala de aula?

d) Explique um benefício e um problema quando se utiliza **Herança** e **Composição** de classes em sistemas OO. Destes dois, qual é o mais recomendado? Porque? Exemplifique

A) Esse padrão vai ser usado em situações em que a gente tem uma classe e essa classe ela possui um conjunto de estados e pra cada estado eu preciso executar uma logica especifica dentro da classe.

Esse padrão me permite mudar o comportamento de um objeto quando seu estado interno é alterado (aparenta ser uma mudança de classe), então quando eu mudo o estado um atributo dentro desse objeto, esse objeto tem um comportamento completamente diferente, ele muda de comportamento.

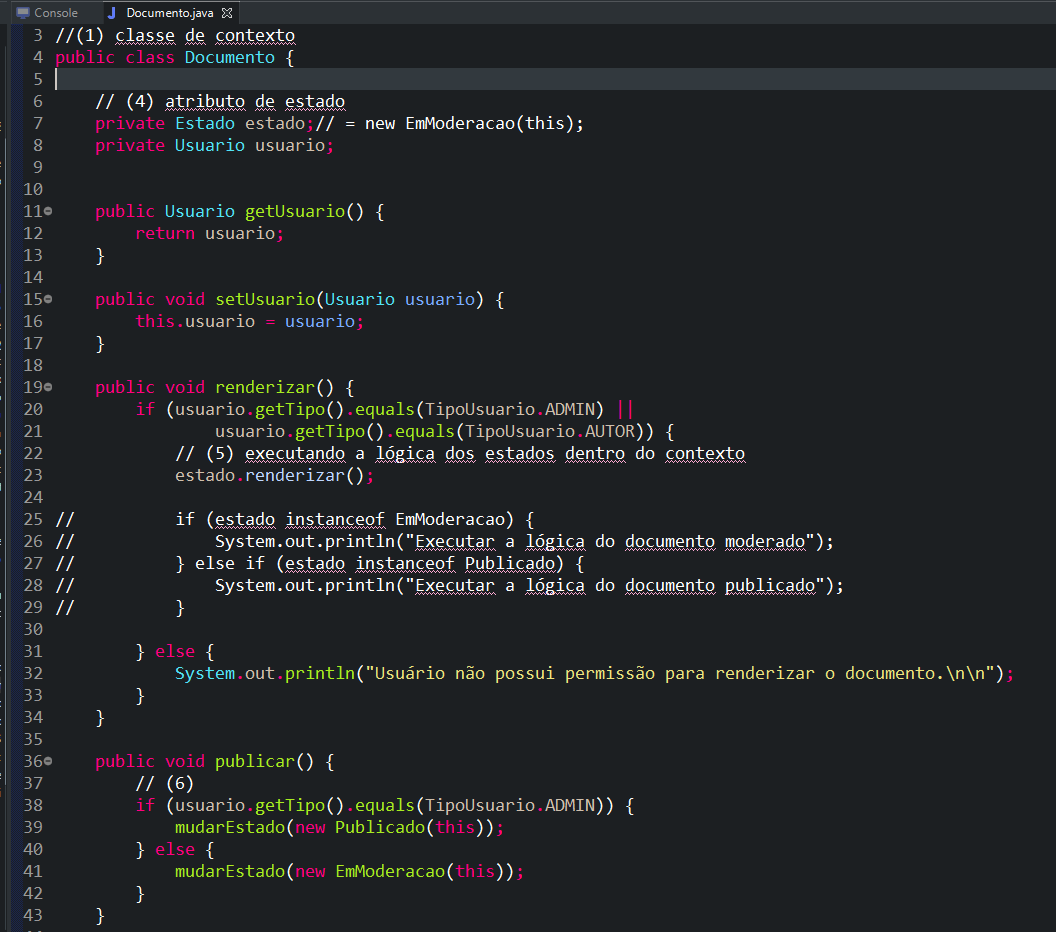
Isso é possível através do estado, temos uma interface state que o contexto vai conter e o state é uma interface com varias implementações especificas, onde cada estado eu vou ter um comportamento especifico daquela operação, então quando eu mudo o estado dentro do contexto eu estou mudando a forma  como o objeto vai se comportar.

Ex: Classe Documento: Ao invés do documento ter todo o código de renderização e publicação, a gente delegou a renderização e publicação pra classes de estado.  A gente tem renderizar que vai repassar pro estado, no caso aqui se o usuário for um admin ou se o usuário for autor  a gente pode renderizar, se não o usuário não tem permissão.

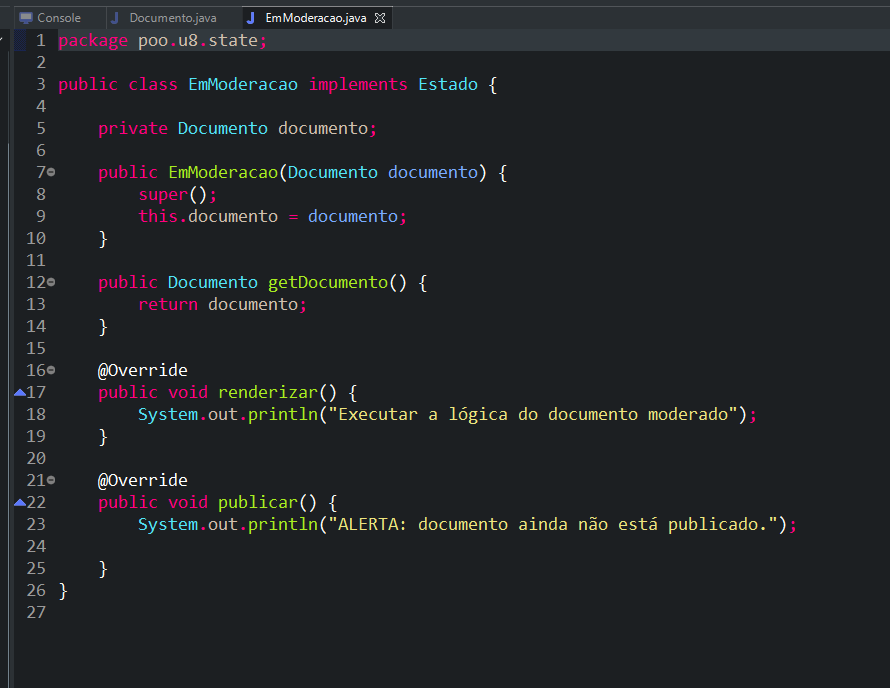
Pra publicar a gente também faz uma verificação se o usuário for um administrador eu consigo publicar, se não vai mudar o estado em moderação

E eu tenho um método perfeitamente pra efetuar o set do estado dentro do documento

Esse documento é um objeto zumbi, porque quem de fato controla o estado desse documento não é documento em si, mas sim os estados que são colocados na variável do estado



Ex de um possível estado:



B) O problema que esse padrão resolve: Reduzir dependências caóticas entre objetos, restringindo a comunicação entre objetos, forçando-os a colaborarem entre si somente via objeto mediador, então a única forma desses objetos interagirem entre si é através justamente do mediador, sem o mediador a gente tem classes, objetos dependendo de vários outros objetos não existe uma forma organizada de manter, de construir essas dependências, então a ideia é termos um mediador  pra controlar a comunicação entre esses objetos  fazer com que todo mundo passe sempre através do mediador pra resolver esse conflito.

Componentes para aplicar esse padrão: A gente tem o colega que é uma interface, temos as implementações desses colegas e temos um mediador, todos os objetos vão ter um referencia pro mediador, o mediador ele vai fazer a mediação/dialogo entre colegas que é justamente que a implementação da interface vai fazer, ele vai mediar ou dialogar entre dois colegas ou entre vários colegas do sistema, ele é quem vai fazer o meio de campo.

Colegas não conversam entre si, existe conversa entre conversa entre colegas e mediador no meio de campo, então o colega  faz a mediação, pega o estado, pega a ação, colega 2 quer conversa com o colega 1 então ele inicia a conversa com o mediador, pega o estado e o mediador aciona as operações de cada colega, ele é responsável pela execução da ação de cada um dos colegas.

C) **Acoplamento** é o que a gente quer que seja **baixo** e **coesão** é o que a gente quer que seja **alto**, ter acoplamento é quando por exemplo o cliente tem contato direto com os atributos de uma determinada classe, caso modifique algum atributo dessa determinada classe, a gente teria que alterar no cliente também, com o acoplamento alto acaba criando interdependências fortes, se alterar ne um tem que alterar no outro pois ele depende do outro, já a coesão é o que faz sentido com o objetivo que a classe propõem. A relação que esses dois conceitos tem com os padrões é para solucionar o problema que acontece com frequência: que é o próprio alto acoplamento --> Muita dependência entre classes é o chamado código espaguete tudo interligado entre si, muitas dependências entre classes, esse é um dos principais problemas que os padrões de objeto tentam resolver, especialmente os padrões de criação, com isso, resolvendo esse problema de alto acoplamento acaba que cria uma alta coesão, cada classe vai executar o seu objetivo, tendo o **SRP single responsibility principle**(Principio de responsabilidade única).

D)Herança: Beneficio: Um dos benefícios da Herança é que ela captura o que é comum e o isola daquilo que é diferente, além disso, a herança é vista diretamente no código até mesmo devido a sua natureza estática.

Problema: O fraco encapsulamento entre classes e subclasses e o forte acoplamento entre elas onde ao mudar uma superclasse pode afetar todas as subclasses além de violar o princípio básico de projeto OO em que devemos ter sempre um baixo acoplamento entre as classes.

Composição: Beneficio: 1º Menor dependência, 2º  o comportamento pode ser escolhido em tempo de execução em vez de estar amarrado em tempo de compilação, 3º Um bom encapsulamento onde os detalhes internos dos objetos instanciados não são visíveis.

Problema: A grande desvantagem é que a aplicação fica mais difícil de entender, pois ela esta mais dinâmica e parametrizada, e uma aplicação mais estática é mais fácil de entender.

qual é o mais recomendado? Composição, porque 1º a composição ela é muito utilizada nos padrões de projetos vistos em sala de aula, isso prova que a composição tem uma grande utilidade e desenvolve projetos melhores 2º Quando se usa herança acaba que traz um forte acoplamento e os padrões tentam evitar isso de todo o jeito, 3º os objetos podem assumir mais de um comportamento, é mais dinâmico, já com herança o objeto é mais estático, 4º Os projetos se toram mais reutilizáveis, 5º e por fim, descarta os problemas envolvendo a herança como o acoplamento entre as classes, que como foi escrito antes, é o que os padrões tentam evitar de todo jeito, favorecendo mais reusabilidade e produtividade