

Exercício 4

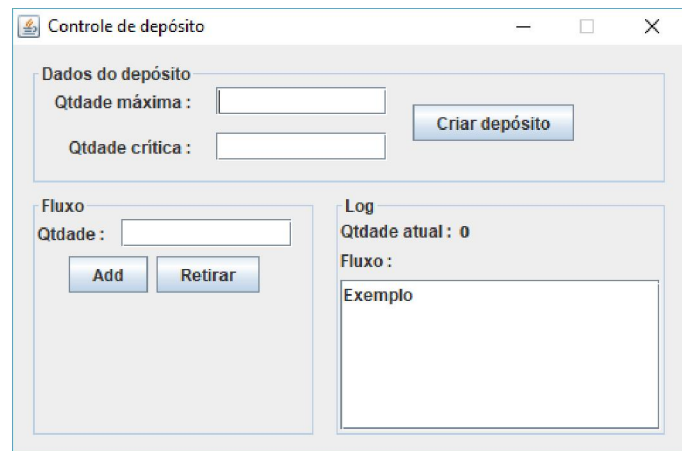
A empresa Têxtil S/A tem um depósito para armazenar seu estoque de malhas. Obviamente, esse depósito tem um limite físico que aceita uma quantidade máxima de malha. Ao mesmo tempo, a empresa estabelece uma quantidade crítica, que se refere à quantidade mínima para manter seus pedidos. Se o depósito atingir essa quantidade os compradores precisam ser avisados urgentemente.

Deposito
- qtdadeAtual : int - qtdadeCritica : int - qtdadeMax : int
+ Deposito(qtdadeCritica : int, qtdadeMax : int) + add(qtdade : int) : void + retirar(qtdade : int) : int

Para representar esse depósito precisamos de uma classe conforme apresentada na figura ao lado. No método `add()` é preciso consistir se a quantidade que estiver sendo passada como parâmetro, acrescida da quantidade atual estourar a quantidade máxima. Nesse caso, a quantidade atual fica igual à quantidade máxima e é disparada uma exceção com a mensagem "Tentativa de armazenar mais que o permitido". O método `retirar()` recebe como parâmetro a quantidade que

pretende consumir do depósito, e retorna a quantidade real que ele conseguiu fornecer. Se a quantidade solicitada for maior que a existente, então retorna somente a quantidade que estiver em depósito.

Baixe a aplicação `observer4` do Moodle. Ela contém uma janela como apresentada na figura ao lado. Através do painel "Dados do depósito" o usuário cria um depósito (a janela deve suportar somente um objeto `Deposito`). Os campos deste painel devem ser utilizados para passar os parâmetros no construtor do objeto. Guarde o objeto em um atributo da janela. O painel "Fluxo" permite adicionar e retirar quantidades do depósito criado. Para o botão "Add", caso ocorra uma exceção, a mensagem deve ser apresentada em um `JOptionPane`. Para o botão "Retirar", a quantidade que conseguiu tirar do depósito deve ser apresentada em um `JOptionPane`. Implemente até aqui, para depois continuar a leitura do enunciado.



O painel "Log" mostra todas as operações realizadas no depósito, assim como a sua quantidade atual. Quem deve povoar essas informações deve ser o próprio `Deposito`, porém, ela não pode ter uma relação direta com a janela, ou seja, não pode haver um atributo, parâmetro ou variável que se refira à classe `ControleDeposito`.

Para resolver esse problema, aplique o padrão **Observer**. Faça uma interface (que chamaremos de Observador ou Listener), e `Deposito` tenha uma associação de agregação com essa interface. Adicione os métodos para adicionar e remover os observadores. Na interface precisam haver três métodos para (1) receber como parâmetro a quantidade atual do `Deposito`, (2) para receber como parâmetro a quantidade que conseguiu adicionar, e (3) para receber como parâmetro a quantidade que conseguiu retirar. A janela (`ControleDeposito`) realiza essa interface. A janela, ao realizar essa interface, precisará ter esses três métodos. Implemente na janela para atualizar os componentes do Log de acordo com o método invocado. Se foi adicionada uma quantidade, então mostra no `JList` "Adicionado nnn", onde "nnn" é a quantidade adicionada. Se foi retirado então mostra "Retirado nnn". No código fonte da janela já existe exemplificada uma classe de modelo que armazena os dados do `JList` que você pode refazer. No construtor, métodos para adicionar e retirar, a classe `Deposito` precisa notificar seus observadores conforme a informação que precisa ser avisada (mudança da quantidade, conseguiu adicionar e conseguiu retirar). Implemente até aqui.

Crie uma nova classe deste `Observer\Listener` que mostra os dados num `JOptionPane`. Instancie dois objetos dessa classe e registre no `Deposito`. Não esqueça que devem ser avisados tanto a janela quanto esses objetos.