

Ficha Prática #08

8.1 Objectivos

Esta Ficha Prática tem como objectivo praticar o desenho de **Diagrama de Máquinas de Estado**.

Uma das possíveis utilizações das máquinas de estado é a modelação do comportamento da interface de uma aplicação (veja a Secção 8.1.1). Não é no entanto a única e esta ficha sobre diversos tipos de utilização.

8.2 Exercícios

Para os exercícios abaixo propostos analise os enunciados e crie os Diagramas de Máquinas de Estado pedidos.

8.2.1 Stack

Considere uma classe `StackBilhete` que implementa os seguintes métodos:

```
public void push(Bilhete b);  
public void pop();  
public Bilhete top();  
public boolean isEmpty();
```

com o comportamento esperado (e bem conhecido) de uma Stack.

Escreva um **Diagrama de Máquina de Estado** que descreva o ciclo de vida da Stack de bilhetes.

8.2.2 Torniquete

Como parte da preparação para o Mundial 2022 foi-lhe pedido que desenvolvesse o software de controlo dos torniquetes de validação de acessos aos estádios. Os

torniquetes possuem braços metálicos que podem impedir ou permitir a passagem de pessoas conforme estejam bloqueados ou não. Possuem também um leitor de bilhetes para validação dos títulos de acesso aos jogos.

A solução que vai propor tem o seguinte modo de funcionamento:

Ao ser ligado o torniquete fica bloqueado. Quando é inserido um bilhete válido o bilhete é retido e o torniquete é desbloqueado, ficando nesse estado até que alguém passe, altura em que bloqueia novamente. Quando é inserido um bilhete inválido, faz soar um alarme e o bilhete é devolvido. O sistema mantém o alarme até o bilhete ser retirado do leitor, altura em que volta a estar bloqueado. Enquanto está desbloqueado o torniquete não aceita bilhetes.

O sistema possuirá também um modo de teste para permitir aos técnicos analisar o funcionamento do torniquete. Quando em modo de teste os técnicos podem realizar dois tipos de testes: testar os braços e testar o leitor de bilhetes. Para testar os braços o técnico pode bloquear e desbloquear o torniquete. Para testar o leitor de bilhetes o técnico insere bilhetes (que poderão ser válidos ou inválidos) que serão sempre devolvidos. No caso do bilhete ser inválido o sistema soa o alarme até o técnico retirar o bilhete. Os dois tipos de teste podem ser executados em paralelo.

O modo de teste pode ser activado em qualquer altura. Quando sai de modo de teste o sistema fica na situação em que estava no momento em que o teste foi iniciado. O sistema pode ser desligado em qualquer altura.

- a) Escreva um **Diagrama de Máquina de Estado** que modele o comportamento do torniquete.
- b) Agora que especificou o comportamento descrito, considera necessário alterar algum aspecto da proposta de modo a garantir o correcto funcionamento do sistema?.

8.2.3 Relógio

Modele o modo de funcionamento de um relógio com alarme. O relógio deverá possuir três botões (modo, hora, minutos) e um comutador (activo/inactivo) para o alarme.

- O botão modo permite de percorrer de forma sequencial os diferentes modos do relógio:
 - Apresentação da hora actual (quer o alarme esteja activo ou não).

- Regulação da hora actual
 - Regulação da hora do alarme
- O comutador alarme permite activar e desactivar o alarme. O relógio toca quando o alarme está activo, o relógio está em modo de apresentação da hora, e a hora actual é igual à hora do alarme.
 - Os outros dois botões (hora e minutos) permitem incrementar as horas e os minutos. Quando o modo é regulação da hora actual, eles alteram a hora actual do relógio. Quando o modo é regulação da hora do alarme, eles alteram a hora a que o alarme vai tocar. No modo de apresentação da hora actual os botões não têm qualquer efeito.

Escreva o **Diagrama de Máquina de Estado** que descreve o modo de funcionamento do relógio.

8.2.4 Encomenda

Considere a seguinte interface Java:

```
public interface Encomenda {  
    public boolean satisfazLEnc(ArrayList<LinhaEnc> linhas);  
    public boolean satisfazLEnc();  
    public void envia();  
    public void cancela();  
    public void adiciona(ArrayList<LinhaEnc> linhas);  
}
```

Uma classe que implemente a interface Encomenda deverá ter o seguinte comportamento:

Após ser criada uma encomenda fica pendente a aguardar que todas as suas linhas sejam satisfeitas. Quando todas as linhas estiverem satisfeitas, a encomenda passa de pendente a completa.

Existem duas formas de registar que uma (ou mais) linha(s) da encomenda já está/estão satisfeita(s). O método `satisfazLEnc(ArrayList<LinhaEnc> linhas)` assinala a satisfação das linhas passadas como parâmetro. O método `satisfazLEnc()` assinala a satisfação de todas as linhas ainda por satisfazer. Após estar completa a encomenda passa a poder ser enviada.

O envio da encomenda é registado utilizando o método `envia()`. Após ser enviada a encomenda fica registada como expedida.

Em qualquer momento antes do envio é possível cancelar a encomenda (método `cancela()`). Nesse caso ela fica registada como cancelada.

O método `adiciona(ArrayLis<LinhaEnc> linhas)` pode ser utilizado em qualquer encomenda que não cancelada ou expedida para lhe adicionar novas linhas.

Escreva um **Diagrama de Máquina de Estado** que represente o comportamento descrito.