MAC0321 – Exercício Programa Individual – Enfermaria

Marcelo Finger

16 de abril de 2024

Instruções

Nome do projeto java: EP1-NUSP. Sendo X o número da lista e NUSP o seu número usp.
Arquivo de envio: Um ÚNICO projeto java zipado, ou seja, enviar o arquivo EP1-NUSP zipado.

Seguir as seguintes instruções, sob pena de desconto de nota por código confuso:

- Não enviar arquivos soltos.
- Não enviar mais de um projeto java.
- Não enviar um projeto com muitas subpastas fazendo com que dê trabalho para encontrar o exercício.
- Não enviar rascunhos dos exercícios. Envie apenas os arquivos que vocês efetivamente querem que sejam corrigidos para que não corram o risco de corrigirmos os arquivos errados.
- Colocar cada exercício dentro de um pacote diferente (o pacote fica dentro do projeto java).
- Separar a classe funcional da classe de testes deixando-as em arquivos diferentes.

Introdução

Vamos usar um padrão pré-pronto de base para o nosso exercício, que será o de simular uma enfermaria especial de infecções. O sistema deve reagir a eventos médicos, de uma forma a ser determinada mais abaixo. Neste ponto, só precisamos saber que termos vários **eventos**, que deverão disparar **ações**. Agora vamos ver como gerenciar estes conceitos: eventos e ações.

Para isso vamos usar um **padrão de projeto** de software. Padrões de projeto são soluções típicas para problemas comuns em design de software. Cada padrão é como um projeto que você pode personalizar para resolver um determinado problema de design em seu código.

Neste exercício, vamos usar um padrão de projeto básico, chamado de *controlador* (*controller*). Este padrão visa resolver o seguinte problema muito recorrente em programação: Qual elemento do sistema deve ser responsável por gerenciar eventos do sistema? E o padrão indica uma classe que representa o responsável por reagir aos evento, o controlador.

Padrões de projeto têm uma apresentação abstrata, para serem implementados em qualquer linguagem orioentada a objeto. Para fazer com que eles tenham uma aparência mais concreta, observe a forma utilizada por Bruce Eckel[1] para escrever um esqueleto de programa que controla uma *Greenhouse* (estufa para plantas). Neste exercício estaremos interessados nos controlador de eventos, e a classe *Greenhouse* servirá apenas de inspiração sobre como podemos fazer um simulador.

A versão do Eckel é feita para uma versão antiga, por isso diponibilizamos uma versão mais atualizada[2]. Primeiro foi criada uma classe abstrata que serve de base para os eventos. Esta base é responsável pela definição de quais métodos devem estar disponíveis para os eventos:

[1] Thinking in Java, disponível na internet no endereço: `www.bruceeckel.com` ou `https://github.com/media-lib/prog_lib/blob/master/java/Bruce%20Eckel%20-%20Thinking%20in%20Java%204th%20Edition.pdf`

[2] `https://gist.github.com/migmc/3993f0a5d511259c8f5a3db9fc4bd26f`

```java
abstract public class Event {
    private long evTime;

    public Event(long eventTime) {
        evTime = eventTime;
    }

    public boolean ready() {
        return System.currentTimeMillis() >= evTime;
    }

    abstract public void action();
    abstract public String description();
}
```

Vemos claramente que nesta classe existem 4 métodos, sendo dois deles abstratos (`action` e `description`). No método `Event` (construtor), o instante a partir do qual o evento pode ser disparado é determinado. O método `ready` apenas retorna verdadeiro caso o objeto evento em questão está "pronto".

A partir desta classe `Event`, uma lista ligada para armazenar e dar acesso a vários objetos do tipo `Event` é criada, seguida de uma classe para gerenciar as ações disparadas pelo eventos:

```java
import java.util.LinkedList; // See
    https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/LinkedList.html

public class Controller {
    // See more about generics here:
        https://docs.oracle.com/javase/tutorial/java/generics/types.html
    private LinkedList<Event> events = new LinkedList<Event>();
    public void addEvent(Event c) { events.add(c); };
    public void run() {
        while (!events.isEmpty()) {
            Event e = events.pop();
            if (e.ready()) {
                e.action();
                System.out.println(e.description());
            } else {
                events.add(e);
            }
        }
    }
}
```

A classe `Controler` gerencia a lista de eventos. Seus métodos permitem a adição de um elemento, e a varredura dos eventos, enquanto existirem eventos disponíveis, ativando as ações de cada evento da lista.

Finalmente na classe `GreenhouseControls` serão criados eventos (como classe privada) para controlar uma estufa.

```java
public class GreenHouseControls extends Controller {
    private boolean light = false;
    private boolean water = false;
    private String thermostat = "Day";

    private class LightOn extends Event {
        public LightOn(long eventTime) {
            super(eventTime);
        }
```

```
10
11          public void action() {
12              light = true;
13          }
14
15          public String description() {
16              return "Light is on!";
17          }
18      }
19
20      private class LightOff extends Event {
21          public LightOff(long eventTime) {
22              super(eventTime);
23          }
24
25          public void action() {
26              light = false;
27          }
28
29          public String description() {
30              return "Light is off!";
31          }
32      }
33
34      private class WaterOn extends Event {
35          public WaterOn(long eventTime) {
36              super(eventTime);
37          }
38
39          public void action() {
40              water = true;
41          }
42
43          public String description() {
44              return "Water is on!";
45          }
46      }
47
48      private class WaterOff extends Event {
49          public WaterOff(long eventTime) {
50              super(eventTime);
51          }
52
53          public void action() {
54              water = false;
55          }
56
57          public String description() {
58              return "Water is off!";
59          }
60      }
61
62      private class ThermostatDay extends Event {
63          public ThermostatDay(long eventTime) {
64              super(eventTime);
65          }
66
```

```
67          public void action() {
68              thermostat = "Day";
69          }
70
71          public String description() {
72              return "Thermostat on day setting";
73          }
74      }
75
76      private class ThermostatNight extends Event {
77          public ThermostatNight(long eventTime) {
78              super(eventTime);
79          }
80
81          public void action() {
82              thermostat = "Night";
83          }
84
85          public String description() {
86              return "Thermostat on night setting";
87          }
88      }
89
90      private int rings;
91      private class Bell extends Event {
92          public Bell(long eventTime) {
93              super(eventTime);
94          }
95
96          public void action() {
97              System.out.println("Bing!");
98              if(--rings > 0) {
99                  addEvent(new Bell(System.currentTimeMillis() + 2000));
100             }
101         }
102
103         public String description() {
104             return "Ring bell";
105         }
106     }
107
108     private class Restart extends Event {
109         public Restart(long eventTime) {
110             super(eventTime);
111         }
112
113         public void action() {
114             long tm = System.currentTimeMillis();
115             // Instead of hard-wiring, you could parse configuration
116             // information from a text file here
117
118             rings = 5;
119             addEvent(new ThermostatNight(tm));
120             addEvent(new LightOn(tm + 1000));
121             addEvent(new LightOff(tm + 2000));
122             addEvent(new WaterOn(tm + 3000));
123             addEvent(new WaterOff(tm + 8000));
```

```
            addEvent(new Bell(tm + 9000));
            addEvent(new ThermostatDay(tm + 10000));
            addEvent(new Restart(tm + 20000));
        }

        public String description() {
            return "Restarting system";
        }
    }

    public static void main(String[] args) {
        GreenHouseControls gc = new GreenHouseControls();
        long tm = System.currentTimeMillis();
        gc.addEvent(gc.new Restart(tm));
        gc.run();
    }
}
```

Exercício 1

Voce deverá traduzir as classes `Event` e `Controler`, de forma a passar nos testes fornecidos na classe `TestaEx1`, juntamente com a classe `EventoSimples`, usada apenas para teste. Inserir suas classes em `usp.mac321.ep1.ex1`

Exercício 2

Considere uma enfermaria especial em que um médico acompanha pacientes acometidos de uma infecção misteriosa. De tempos em tempos, o médico verifica os sinais vitais dos pacientes, já sabendo que esta infecção se manifesta gerando picos de temperatura elevada. Se o paciente fica com a temperatura acima de 41°C por mais de uma hora ele vem a óbito.

Nesta segunda parte do exercício você deve produzir classes para médico, paciente e droga, inserindo-as no pacote `usp.mac321.ep1.ex2`. O médico é caracterizado pela frequência com que monitora os pacientes, pelo conjunto de drogas que ele tem a sua disposição e por um algoritmo através do qual ele decide qual droga administrar para cada paciente, e a duração de cada administração; o médico pode ter outros atributos que você julgar conveniente.

Cada *Droga d* possui uma velocidade de redução da temperatura (v_T^d) e uma velocidade de redução da concentração de uma proteína anti-coagulação (PAC) no sangue (v_c^d); estes elementos serão discutidas mais a seguir.

Nesta enfermaria, os sinais vitais que o médico monitora em um *Paciente* são a temperatura e a concentração de PAC no sangue. Cada paciente é caracterizado por uma temperatura basal (um valor é entre 36 e 37 graus Celsius), uma concentração basal de PAC no sangue (um valor entre 100 e 150 mil unidades por mm^3), um valor do aumento de temperatura causada por um surto de infecção (que deve ser um valor de no mínimo 5°C), a frequência com que os surtos de infecção ocorrem, e a velocidade aumento de PAC uma vez cessada a medicação (a ser discutido mais adiante). O paciente deve conter, no mínimo, os seguintes métodos:

- Um método que recebe um instante de tempo e informa a temperatura do paciente naquele instante;
- Um método que recebe um instante de tempo e informa a concentração de PAC no sangue do paciente naquele instante;
- Um método que informa se o paciente está vivo. Se o paciente não estiver vivo, o valor de seus sinais vitais deverá ser negativo.

No mesmo pacote, deve-se criar uma classe de testes que teste os métodos básicos das classes Médico, Droga e Paciente.

Exercício 3

Vamos usar o controlador de eventos para simular os acontecimentos na enfermaria especial. Neste exercício vamos simular um médico incompetente que, por ignorância, falta de recursos ou má fé, não administra nenhum medicamento aos pacientes, apenas monitora os seus sinais vitais até que, inevitavelmente, este venha a falecer.

No pacote `usp.mac321.ep1.ex3`, implementar uma classe `Simulador1` derivada de `Controlador` do Exercício 1. Esta classe deve ter uma série e eventos, que criam o médico e paciente, inicia o surto de infecção no paciente, insere os monitoramentos periódicos do médico e registra os sinais vitais do paciente a cada monitoração através do método descrição dos eventos. O médico deve monitorar os sinais vitais a cada 10 minutos.

Note que, numa simulação, um segundo no mundo real pode representar vários minutos simulados, e isso será um parâmetro do seu simulador. Sugerimos que cada 10 ms no mundo real equivalha a 1 min simulado.

Vamos simular a enfermaria com só um médico e só um paciente, até o instante em que o paciente morre. No mesmo pacote, você deve criar uma classe de teste com um único método que verifica que o simulador imprime exatamente o que se espera. Por exemplo, o seu simulador pode imprimir

```
0 min
Médico criado
Paciente criado
Médico consulta temperatura: 36.5
Médico consulta concentração: 110000
5 min
Paciente inicia surto infecioso
10 min
Médico consulta temperatura: 41.5
Médico consulta concentração: 110000
...
70 min
Médico verifica óbito do paciente
Simulação terminada
```

e o seu teste tem que verificar que o programa imprime exatamente isso.

Exercício 4

No pacote `usp.mac321.ep1.ex4`, implementar uma classe `Simulador2`, com novos eventos que permitam a um médico bem intencionado aplicar drogas aos pacientes.

Para tratar os pacientes o médico está testando uma nova droga experimental de administração endovenosa e de efeito muito rápido, cujo nome secreto nesta fase de testes é clorokaka (CKK). Apesar do rápido efeito desta droga, notou-se que em pacientes infectados há formação de coágulos sanguíneos (trombos). Descobriu-se que existe uma proteína anticoagulante (PAC) cuja concentração é reduzida pela clorokaka, o que pode levar a óbito um paciente que fique por mais de trinta minutos com um limiar muito baixo do nível de PAC. Notou-se também que, uma vez que a medicação é suspensa, a quantidade de PAC volta a aumentar. Assim sendo, um tratamento experimental procede a administração da droga por intervalos de tempo limitados, para tentar manter o paciente com os sinais vitais dentro de uma faixa razoável. Nesta simulação, os sinais vitais importantes são a temperatura e a concentração de PAC no sangue dos pacientes. O médico deve monitorar estes valores em cada paciente a cada 10 minutos.

A temperatura de um paciente sob efeito da droga varia com o tempo. Ela sofre um pico de elevação no instante do surto infeccioso e sofre uma redução pelo efeito da administração de uma droga. Considere a seguinte equação que descreve a queda de temperatura T em um paciente sob o efeito da droga d =CKK:

$$T = T_{basal} + T_s e^{-v_T^d (t - t_d)} \tag{1}$$

onde T_{basal}: temperatura basal do corpo do paciente
T_s: acrescimo de temperatura causado no surto da infecção
v_T^d: velocidade de redução da temperatura com a droga d
t_d: instante inicial da administração da droga

A concentração de PAC no sangue também varia com o tempo. Ela começa a cair com administração da droga e retorna aos níveis basais com a interrupção da administração da droga. Considere a seguinte equação que descreve a queda de da concentração c de PAC no sangue de um paciente sob o efeito da droga d =CKK:

$$c = c_{basal} e^{-v_c^d (t - t_d)} \tag{2}$$

onde c_{basal}: concentração basal de PAC no sangue do paciente
v_c^d: velocidade de redução da concentração com a droga d
t_d: instante inicial da administração da droga

Já o retorno da concentração de PAC no sangue do paciente quando cessa a medicação segue a seguinte equação:

$$c = c_{basal} - (c_{basal} - c_0) e^{-v_r (t - t_d)} \tag{3}$$

onde c_{basal}: concentração basal de PAC no sangue do paciente
c_0: concentração inicial de PAC no sangue do paciente
v_r: velocidade de recuperação da concentração de PAC no sangue do paciente
t_d: instante da parada da administração da droga

Nas equações (1), (2) e (3), o tempo é medido em minutos. A temperatura do paciente acima de $41°C$ por uma hora leva a óbito. A concentração de PAC no sangue abaixo de 50.000 unidades por mm^3 também leva a óbito em meia hora. E, por fim, temos os valores basais de temperatura entre $36°C$ e $37°C$ e os de concentração de PAC entre 100.000 e 150.000 unidades por mm^3. O pico de temperatura causado pelo surto da infecção é de, no mínimo, um acréscimo de $5°C$ à temperatura basal. A verificação dos sinais vitais do paciente e feita a cada 10 minutos.

A droga CKK possui velocidade de redução de temperatura de $27 \times 10^{-3}\text{min}^{-1}$ e velocidade de redução da concentração de PAC de $54 \times 10^{-3}\text{min}^{-1}$. O paciente tem uma velocidade de recuperação da concentração de PAC no sangue de $38 \times 10^{-3}\text{min}^{-1}$.

Os surtos de infecção dos pacientes ocorrem a cada 2 horas, com imediato aumento da temperatura.

Pede-se a apresentação de uma simulação de acordo com o comportamento acima. A simulação envolve apenas um médico e um paciente, e ela termina com o óbito do paciente ou após 6h de monitoração. Você deve criar novas versões da classe Médico e Paciente para esta simulação e inseri-las no módulo correspondente.

Não é necessário criar testes neste caso, apenas executar a simulação através de um método `main` na classe `Simulador2`.

A impressão da simulação deve aparecer como um comentário ao final do arquivo que contém a classe `Simulador2`.