MAC0321 – Exercício Programa Individual – Enfermaria

Marcelo Finger

27 de maio de 2021

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.

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*sefvirá apenas de inspiração de como podemos fazer um simulador

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  package c07.controller;
2
3  abstract public class Event {
4    private long evtTime;
5    public Event(long eventTime) {
6      evtTime = eventTime;
7    }
8    public boolean ready() {
9      return System.currentTimeMillis() >= evtTime;
10   }
11   abstract public void action();
12   abstract public String description();
13 } ///:~
```

[1] Thinking in Java, disponível na internet no endereço: `www.bruceeckel.com`

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 classe para controlar vários objetos do tipo Event é criada:

```
1  package c07.controller;
2
3  // This is just a way to hold Event objects.
4  class EventSet {
5    private Event[] events = new Event[100];
6    private int index = 0;
7    private int next = 0;
8    public void add(Event e) {
9      if(index >= events.length)
10       return; // (In real life, throw exception)
11     events[index++] = e;
12   }
13   public Event getNext() {
14     boolean looped = false;
15     int start = next;
16     do {
17       next = (next + 1) % events.length;
18       // See if it has looped to the beginning:
19       if(start == next) looped = true;
20       // If it loops past start, the list
21       // is empty:
22       if((next == (start + 1) % events.length)
23          && looped)
24         return null;
25     } while(events[next] == null);
26     return events[next];
27   }
28   public void removeCurrent() {
29     events[next] = null;
30   }
31 }
32
33 public class Controller {
34   private EventSet es = new EventSet();
35   public void addEvent(Event c) { es.add(c); }
36   public void run() {
37     Event e;
38     while((e = es.getNext()) != null) {
39       if(e.ready()) {
40         e.action();
41         System.out.println(e.description());
42         es.removeCurrent();
43       }
44     }
45   }
46 } ///:~
```

Na classe `EventSet` vemos que existem três membros privados: `index` que corresponde à quantidade de eventos no vetor de eventos `Event`, e `next` que corresponde ao evento em questão. Existem também métodos para adicionar um evento ao vetor, pegar o próximo evento (não nulo) do vetor e remover o evento em questão.

A classe `Controler` gerencia um elemento da classe `EventSet`. Seus métodos permitem a adição de um

elemento, e a varredura dos eventos, enquanto existirem eventos disponíveis.

Finalmente na classe `GreenhouseControls` serão criados eventos para controlar uma estufa.

```
package c07.controller;

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);
    }
    public void action() {
      // Put hardware control code here to
      // physically turn on the light.
      light = true;
    }
    public String description() {
      return "Light is on";
    }
  }
  private class LightOff extends Event {
    public LightOff(long eventTime) {
      super(eventTime);
    }
    public void action() {
      // Put hardware control code here to
      // physically turn off the light.
      light = false;
    }
    public String description() {
      return "Light is off";
    }
  }
  private class WaterOn extends Event {
    public WaterOn(long eventTime) {
      super(eventTime);
    }
    public void action() {
      // Put hardware control code here
      water = true;
    }
    public String description() {
      return "Greenhouse water is on";
    }
  }
  private class WaterOff extends Event {
    public WaterOff(long eventTime) {
      super(eventTime);
    }
    public void action() {
      // Put hardware control code here
      water = false;
    }
    public String description() {
      return "Greenhouse water is off";
```

```
    }
  }
  private class ThermostatNight extends Event {
    public ThermostatNight(long eventTime) {
      super(eventTime);
    }
    public void action() {
      // Put hardware control code here
      thermostat = "Night";
    }
    public String description() {
      return "Thermostat on night setting";
    }
  }
  private class ThermostatDay extends Event {
    public ThermostatDay(long eventTime) {
      super(eventTime);
    }
    public void action() {
      // Put hardware control code here
      thermostat = "Day";
    }
    public String description() {
      return "Thermostat on day setting";
    }
  }
  // An example of an action() that inserts a
  // new one of itself into the event list:
  private int rings;
  private class Bell extends Event {
    public Bell(long eventTime) {
      super(eventTime);
    }
    public void action() {
      // Ring bell every 2 seconds, rings times:
      System.out.println("Bing!");
      if(--rings > 0)
        addEvent(new Bell(
          System.currentTimeMillis() + 2000));
    }
    public String description() {
      return "Ring bell";
    }
  }
  private class Restart extends Event {
    public Restart(long eventTime) {
      super(eventTime);
    }
    public void action() {
      long tm = System.currentTimeMillis();
      // Instead of hard-wiring, you could parse
      // configuration information from a text
      // file here:
      rings = 5;
      addEvent(new ThermostatNight(tm));
      addEvent(new LightOn(tm + 1000));
      addEvent(new LightOff(tm + 2000));
```

```
112       addEvent(new WaterOn(tm + 3000));
113       addEvent(new WaterOff(tm + 8000));
114       addEvent(new Bell(tm + 9000));
115       addEvent(new ThermostatDay(tm + 10000));
116       // Can even add a Restart object!
117       addEvent(new Restart(tm + 20000));
118     }
119     public String description() {
120       return "Restarting system";
121     }
122   }
123   public static void main(String[] args) {
124     GreenhouseControls gc =
125       new GreenhouseControls();
126     long tm = System.currentTimeMillis();
127     gc.addEvent(gc.new Restart(tm));
128     gc.run();
129   }
130 } ///:~
```

Exercício 1

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

Exercício 2

Em 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, sendo que 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 `br.ime.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 PAC no sangue (v_c^d); que serão discutidas mais a seguir.

Nesta enfermaria, os sinais vitais que o médico monitora em um *Paciente* são a temperatura e a concentração de uma proteína anti-coagulação (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, 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 uns instante de tempo e informa a temperatura do paciente naquele instante;
- Um método que recebe uns 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 `br.ime.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, incia o surto de infecção no paciente, contem 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 um médico e 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 `br.ime.usp.mac321.ep1.ex4`, implementar uma classe `Simulador2`, com novos eventos que agora 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 é produzido como efeito colateral a formação de coágulos sanguíneos (trombos). Descobriu-se que existe uma proteína anticoagulante (PAC) cuja concentração é afetada pela medicação de tal forma que a diminuição da quantidade de PAC pode levar ao 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_T^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 acrescimo 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`.