Universidade Federal de São Carlos

Trabalho 3
Paradigmas de Linguagens de Programação

Nome: Julia de Moura Caetano – R.A.: 619655
Turma A

Para realizar este trabalho, foi utilizada a ferramenta IntelliJ, versão 2017.1.

Códigos-fonte e descrição das funcionalidades

Sala.java

```
public class Sala
{
    private String id, localreserva;
    private int capacidade;
    private boolean projetor, livre;

      Sala(String _id, String _localreserva, int _capacidade, boolean _proj,
boolean _livre)
    {
        id = _id;
        localreserva = _localreserva;
        capacidade = _capacidade;
        projetor = _proj;
        livre = _livre;
    }

    public String getId(){return id;}

    public void libera(){livre = true;}
    public void reserva(){livre = false;}
    public boolean getEstado(){return livre;}
    public boolean temProjetor(){return projetor;}
    public int getCapacidade(){return capacidade;}
}
```

Esta classe representa uma sala e todos os elementos necessários para torná-la apta para reservas: id, local, capacidade, se tem (assume valor verdadeiro) ou não (assume valor falso) projetor, e se está ou não livre.

AulasTeoricas.java

```
public class AulasTeoricas
{
    ArrayList<Sala> salas = new ArrayList<Sala>(5);

    public void geraSalas(int _num)
    {
        for(int K = 0; K != _num; K++)
        {
            Random n = new Random();

            boolean proj;
            if (n.nextInt() % 2 == 0)
                proj = true;
            else
                proj = false;

               Sala s = new Sala("" + n.nextInt(200), "UFSCar", n.nextInt(61),
proj, true);
            salas.add(s);
        }

    }
```

```
    public Sala verificaSalaLivre(boolean _proj, int _cap)
    {
        for(Sala s: salas)
        {
            if(s.temProjetor() == _proj && _cap <= s.getCapacidade())
            {
                if (s.getEstado())
                    return s;
            }
        }
        return null;
    }

    public synchronized void liberar (Sala s) throws InterruptedException
    {
        s.libera();
        System.out.println("Sala "+s.getId()+" liberada!");
        notify();
    }

    public synchronized Sala reservar(String pdacesso, boolean _proj, int _cap)
            throws InterruptedException
    {
        Sala s;
        //Random t = new Random();
        while((s = verificaSalaLivre(_proj,_cap)) == null)
            wait();
        s.reserva();
        System.out.println("Sala "+s.getId()+" reservada para "+pdacesso);
        notify();
        return s;
    }
}
```

Esta classe é a que realmente realiza as reservas. O primeiro objeto declarado (“salas”) é um vetor dinâmico que guarda objetos do tipo sala, explicado anteriormente.
O primeiro método, “geraSalas”, como o nome diz, gera sala com atributos aleatórios. Primeiro, define-se o valor de proj utilizando um gerador de números aleatórios. Se o número gerado for par, proj assume valor verdadeiro. Caso contrário, assume valor falso. Quando o elemento s a ser criado é declarado, insere-se outro número aleatório com id; o local de reserva é “UFSCar”; o número de cadeiras é gerado de forma aleatória, com o mínimo zero e máximo sessenta; o valor de proj é atribuído à variável que define se há ou não projetor; e toda sala criada como livre.
O próximo método, “verificaSalaLivre”, recebe um booleano _proj e um inteiro _cap. Em um laço que varre todo o vetor “salas”, declarado no escopo da classe, é verificado se o elemento que o laço está tem o mesmo valor de projetor que _proj e se _cap é menor ou igual à capacidade da sala. Se esta condição é verdadeira, é verificado se a sala está livre. Se estiver, a sala “s” é retornado. Senão, é retornado nulo.
O método “liberar” recebe um objeto do tipo Sala e é sincronizado. Este método somente chama o método “libera” do próprio objeto, que muda o atributo “livre” de falso para verdadeiro. Depois, é escrito no console que a sala foi liberada e é notificado que o método acabou sua execução.

O método “reservar” recebe uma string que representa “quem” está requisitando a sala, booleano _proj e inteiro _cap. Enquanto as salas são verificadas e o valor retornado é nulo, espera-se. Quando finalmente uma sala é retornada, esta é reservada. Imprime-se que uma sala foi reservada e o fim da execução do método é notificado e o objeto do tipo Sala que for reservado é retornado.

ReservaThread.java

```
public class ReservaThread extends Thread
{
   /*Declaração do objeto ats, que é uma instância de AulasTeoricas e deve
    * ser enviado do método main */
   private AulasTeoricas ats;

   //Construtor
   public ReservaThread (AulasTeoricas _ats)
   {
      ats = _ats;
   }

   public void run()
   {
      System.out.println("Thread "+this.getName()+" iniciada");

         try {
            Random n = new Random();
            int n1 = n.nextInt(5);

            boolean proj;

            if (n1 % 2 == 0)
               proj = true;
            else
               proj = false;

            Sala s = ats.reservar(this.getName(), proj, n.nextInt(61));
            this.sleep(n.nextInt(1000));
            ats.liberar(s);
         }
         catch(InterruptedException e)
         {
            System.out.print("Ocorreu algum erro :(");
         }
   }
}
```

É declarada uma instância de AulasTeóricas, que vai ser compartilhada por todos as threads declaradas na classe. Essa instância deve ser enviada pelo método que está instanciando um objeto desta classe thread. No método run, é impresso em tela thread corrente, a thread vai reservar uma sala, cujas necessidades também são geradas aleatoriamente. Depois de completo o método reservar, o thread “dorme” por até um segundo, e depois libera a sala que foi reservada.

Main.java

```
import java.util.Scanner;
```

```
public class Main {

    public static void main(String[] args)
    {

       System.out.println("+++++Início+++++");

       Scanner in = new Scanner(System.in);

       System.out.print("Quantas salas devem ser geradas? R.: ");
       int numsalas = Integer.parseInt(in.nextLine());

       AulasTeoricas reservasat = new AulasTeoricas();
       reservasat.geraSalas(numsalas);

       ReservaThread t1 = new ReservaThread(reservasat);
     ReservaThread t2 = new ReservaThread(reservasat);
     ReservaThread t3 = new ReservaThread(reservasat);
     ReservaThread t4 = new ReservaThread(reservasat);

     for(int i = 0; i != 4; i++)
     {
         t1.run();
         t2.run();
         t3.run();
         t4.run();
     }
    }
}
```

Este método tem como objetivo testar o funcionamento do programa. É definido o espaço AulasTeóricas a ser dividido pelas threads, quantas salas serão geradas e as threads em si.

Exemplos de execução

Execução 1

```
+++++Início+++++
Quantas salas devem ser geradas? R.: 100
Thread Thread-0 iniciada
Sala 18 reservada para Thread-0
Sala 18 liberada!
Thread Thread-1 iniciada
Sala 158 reservada para Thread-1
Sala 158 liberada!
Thread Thread-2 iniciada
Sala 158 reservada para Thread-2
Sala 158 liberada!
Thread Thread-3 iniciada
Sala 158 reservada para Thread-3
Sala 158 liberada!
Thread Thread-0 iniciada
Sala 158 reservada para Thread-0
Sala 158 liberada!
Thread Thread-1 iniciada
Sala 18 reservada para Thread-1
Sala 18 liberada!
Thread Thread-2 iniciada
```

Sala 18 reservada para Thread-2
Sala 18 liberada!
Thread Thread-3 iniciada
Sala 18 reservada para Thread-3
Sala 18 liberada!
Thread Thread-0 iniciada
Sala 18 reservada para Thread-0
Sala 18 liberada!
Thread Thread-1 iniciada
Sala 27 reservada para Thread-1
Sala 27 liberada!
Thread Thread-2 iniciada
Sala 18 reservada para Thread-2
Sala 18 liberada!
Thread Thread-3 iniciada
Sala 158 reservada para Thread-3
Sala 158 liberada!
Thread Thread-0 iniciada
Sala 18 reservada para Thread-0
Sala 18 liberada!
Thread Thread-1 iniciada
Sala 158 reservada para Thread-1
Sala 158 liberada!
Thread Thread-2 iniciada
Sala 18 reservada para Thread-2
Sala 18 liberada!
Thread Thread-3 iniciada
Sala 18 reservada para Thread-3
Sala 18 liberada!

Execução 2

+++++Início+++++
Quantas salas devem ser geradas? R.: 10
Thread Thread-0 iniciada
Sala 146 reservada para Thread-0
Sala 146 liberada!
Thread Thread-1 iniciada
Sala 146 reservada para Thread-1
Sala 146 liberada!
Thread Thread-2 iniciada
Sala 146 reservada para Thread-2
Sala 146 liberada!
Thread Thread-3 iniciada
Sala 140 reservada para Thread-3
Sala 140 liberada!
Thread Thread-0 iniciada
Sala 146 reservada para Thread-0
Sala 146 liberada!
Thread Thread-1 iniciada
Sala 140 reservada para Thread-1
Sala 140 liberada!
Thread Thread-2 iniciada

Esta execução não foi terminada pois não houve uma sala na lista que correspondia às demandas da Thread-2.