Sistemas Distribuídos
Exame

1 de Fevereiro de 2016
10/01/2018

II

1.

```
public class Questao implements ;
    int tentativa=0; Questao }
    int id;
    String pergunta;
    String resposta;
    boolean respondido=false;
    RL l=new RL();
}
```

```
public class Biblioteca implements Controlador{
    int idQ=0;
    HashMap<Integer, QuestaoC> questoes;
    RL lock=new RL();
    Condition cond=lock.newCondition();
}
```

```
class QuestaoC {
    public QuestaoC(int id, String pergunta, String resposta){
        this.id=id;
        N.pergunta=pergunta;
        N.resposta=resposta;
    }

    int id(){
        return id;
    }

public boolean podeResponder(){
    l.lock();
    boolean x=((!respondido) && (tentativa<10));  ← *
    l.unlock();
    return x;
}

public String responde(String resp){
    l.lock();
    *
    if(!x){
        tentativa++;
        respondido=resp.equals(resposta);
```

```
        x = respondido;
        l.unlock();
        if(x){
            return("C");
        }
        else{
            return("F");
        }
    }
    else{
        l.unlock();
        return("R");
    }
}
}

class Biblioteca {

public void adiciona(String pergunta, String resposta){
    lock.lock();
    QuestaoC q = new QuestaoC(idQ, pergunta, resposta);
    questoes.put(idQ, q);
    idQ++;
    lock.unlock();
}

public Questao obtem(int id){
    lock.lock();
    Questao q;

    while((q = queroQuestao(id)) == null){
        cond.await();
    }
    lock.unlock();
    return q;
}

public Questao queroQuestao(int id){
    Questao res = null;
```

```
for(QuestaoC q: questoes.values()){
    if((q.id() >= id) && (q.podeResponder())){
        res = (Questao) q;
        break;
    }
}
return res;
}
}
```

```
public enum OpsB{
    Pergunta;
}
```

2.

```
public class TrataCliente implements Runnable{
    Biblioteca b;
    (Set<Integer>) HashMap<Integer, Questao> jaRespondidas; (acho que aqui basta o id!)
    Socket s;

    public TrataCliente(b, s){
        this.b=b;
        this.s=s;
        jaRespondidas= new HashSet<>();
    }

    public void run(){
        OOS out = s.getOS();
        OIS in = s.getIS();

        try{ OpsB op;
            while((op=(OpsB)in.readObject())!=null){
            switch(op){
            Case Pergunta:
                int id= in.readInt();
                Questao q= b.obtem(id);
                while(jaRespondidas.contains(q.id())){
                    q=b.obtem(id);
                }
```

```
out.write(((QuestaoC) q).getPergunta());
out.write("\n");

String resp = in.read

resp = q.responde(resp);

if(resp.equals("R")){
        out.write("Respondida\n");
}
else{
        if(resp.equals("C")){
            out.write("Certa\n");
        }
        else{
            if(resp.equals("F")){
                out.write("Errada\n");
            }
        }
}
jaRespondidas.add(q.id());
out.flush();
break;
default: out.write("Op invalida\n");
         out.flush();
         break;
    }
}
}catch(Exception e){e.printStackTrace();}
finally{ s.shutdownOutput();
         Input();
       }s.close();
}

public class AddQ implements Runnable{
    int tempo=60 000; Biblioteca B;
    public void run(){
```

```
        while(true){
            String[] p= Util.novaPergunta();
            b.adiciona(p[0], p[1]);
        }Thread.sleep(tempo);
    }
}

public class Servidor {

    public static void main(){
        ServerSocket ss= new ServerSocket(9999);
        Biblioteca b= new Biblioteca();
        (new Thread(new AddQ(b))).start();
        while(true){
            Socket s=ss.accept();
            (new Thread(new TrataCliente(b, s))).start();
        }
    }
}
```

I

1. A operação de await(cond, lock) é utilizada porque queremos "esperar" na variável de condição cond até que o predicado que nos teo esperar seja falso. Para invocar esta operação (normalmente) é necessário ter o lock adquirido pelo que quando esta é invocada é necessário que (faça) unlock() liberte o lock e adormeça e que quando acorde o volte a adquirir, daí a necessidade de estar associada ao lock.

3. Em sistemas de objectos distribuídos, um servidor de objectos não oferece um serviço específico, mas mantém um conjunto de objetos que oferecem variados serviços. Tem como principal função gerir um conjunto de objectos e intermediar os pedidos que lhes são realizados. É multi-threaded podendo atribuir uma thread a cada objeto ou a cada invocação.