Atividades sobre Concorrência

Pontuação da Atividade Individual: 35 pontos

10/09/2016

Controle de Versão

Versão	Data	Descrição	Responsável
1.0	10/09/2016	Criação do documento (adicionado as questões de 1 a 2)	Ari
1.1	10/09/2016	Adicionando questões de 3 a 10	Ari
1.2	16/09/2016	Adicionado observação sobre estrutura a estrutura do projeto.	Ari
1.3	16/09/2016	Adicionado questões de 11 a 12	Ari
1.4	20/09/2016	Reduzido o número de questões para 15. Adicionado questões de 13 a 15. Alterado data de entrega.	Ari

As soluções das questões, da mesma forma que o exercício anterior, deverão ser agrupadas em um diretório e depois encaminhadas para o GutHub. Todas as Questões devem ser nomeadas de acordo com o exercício anterior (ex.: q1-node1, q1-node2, q2-node2). Para uma melhor organização, se desejar, pode ainda agrupar os projetos dentro de um diretório com o nome da questão.

Observações:

- **Utilize apenas JDK 1.7.X**
- **Não é permitido utilizar frameworks para resolver problemas de concorrência**
- **Entregar até dia 30/09/2016**
- **Com exceção dos projetos para o GAE, adote a estrutura de projeto Maven para resolver as questões**

Linguagens por aluno

- **DART (Victor)**
- **FORTRAN (Wenstay)**
- **Node.js (Wellington)**
- **Basic (Laerton)**
- **GoLang (Natarajan)**
- **Scala (Aluisio)**
- **Pascal (Dijalma)**

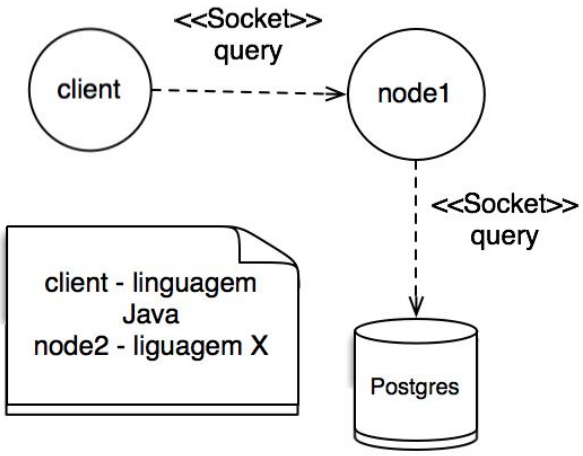
1/15 - Questão

Abra um conta no Heroku e no GAE (Google Appengine) e crie uma aplicação simples de Hello World. Para tanto utilize os seguintes tutoriais:

Para Heroku: https://devcenter.heroku.com/articles/getting-started-with-java#introduction
Para GAE: https://cloud.google.com/appengine/docs/java/quickstart

2/15 - Questão

Considerando a topologia apresentada na figura abaixo, construa uma aplicação que possa realizar consultas em um banco de dados. Para tanto o cliente deve ser escrito na linguagem Java e o nó1 deve ser escrito na linguagem selecionada para cada aluno (sua segunda linguagem).



3/15 - Questão

Implemente o algoritimo apresentado no psudocódigo ao lado utilizando a sua segunda linguagem e informe qual a saída produzida.

```
1
2   //declaração
3   value <- 'B'
4   //procedimento
5   procedure setValue(){
6     value <- 'A'
7   }
8   //thread
9   thread setter {
10    setValue()
11  }
12  //executando
13  start setter;
14  for i <- 0 to 10 {
15    print(value);
16  }
17
```

4/15 - Questão

Implemente o algoritimo apresentado no psudocódigo código ao lado na linguagem Java e informe o resultado da variável x após execução.

```
1
2   //declaração
3   x <- 0
4   y <- 0
5   //procedimento
6   procedure sum(){
7     x <- y + 1
8     y <- x + 1
9   }
10  //thread
11  thread A {
12    sum()
13  }
14  thread B {
15    sum()
16  }
17  //executando
18  for i <- 1 to 10 {
19    start A;
20    start B;
21    print(x);
22  }
23
```

5/15 - Questão

Implemente o mesmo algoritmo da questão anterior na sua segunda linguagem.

6/15 - Questão

Altere o algoritmo da questão 4 e adote *wait()* e *notify()* de forma que o resultado para o valor de x, ao final da execução, seja igual a 21.
(obs.: utilize a linguagem Java)

7/15 - Questão

Altere o algoritmo da questão 4 e adote *synchronize* no procedimento *sum()* e informe o resultado de x ao final da execução.
(obs.: utilize a linguagem Java)

8/15 - Questão

Adote sua segunda linguagem para resolver o mesmo problema da questão 7.

9/15 - Questão

Considere o cenário descrito abaixo e implemente-o em Java.

Em um parque existem N passageiros e um carro em uma montanha russa. Os passageiros, repetidamente, esperam para dar uma volta no carro. O carro tem capacidade para C passageiros, sendo que C, N ; C > 20; C ≈ N e = 0.875.
O carro só pode partir quando estiver cheio. Após dar uma volta na montanha russa, cada passageiro passeia pelo parque de diversões e depois retorna à montanha russa para a próxima volta. Cada volta na montanha russa leva 1ft (fake time = 1s). Cada passeio no parque de diversões leva entre 1ft e 3ft. Após 16ft o parque fecha.

Dicas:

- tanto o carro como os passageiros devem ser representados por threads.

- tanto o carro quanto o passageiro devem seguir as interfaces abaixo apresentadas.

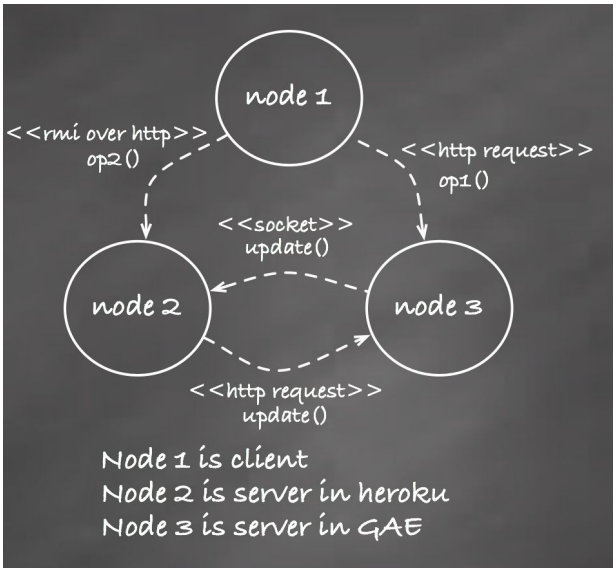
```
24
25  interface Passenger{
26    void getInTheCar(); //entrar no carro
27    void waitRideAway(); //aguardar passeio até finalizar
28    void getOutTheCar(); //sair do carro
29    void rideInThePark(); //passear pelo parque
30  }
31
32  interface Car{
33    void waitFill(); //aguardar encher
34    void takeAWalk(); //dar uma volta
35    void waitGetOutAll(); //aguardar sair todos
36  }
37
```

10/15 - Questão

Resolva a questão 9 adotando a sua segunda linguagem.

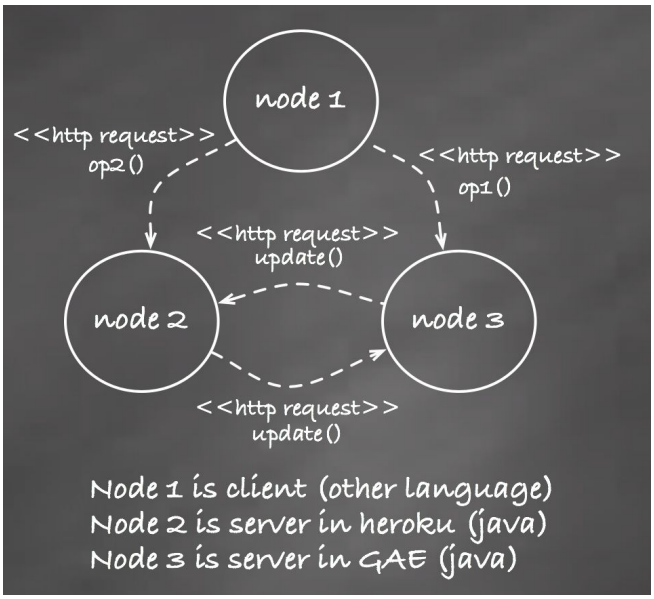
11/15 - Questão

A figura abaixo apresenta a topologia de um sistema distribuído a ser implementado em Java que utiliza comunicação baseada em RMI e adota o protocolo de transporte HTTP. O objetivo do sistema é manter a contagem global do número de execuções das operações op1() e op2(). Implemente este sistema de forma a garantir que os valores no node2 sejam iguais aos do node3. Utilize este código (https://github.com/aristofanio/ag-rmi-in-heroku) como exemplo para executar RMI/HTTP no Heroku.



12/15 - Questão

Implemente a questão anterior substituindo RMI por Socket conforme pode ser visto na figura abaixo. Além disto o cliente (node1) deve ser implementado utilizando sua segunda linguagem.



13/15 - Questão

Usando a técnica de Long Polling implemente um chat estilo UOL Batepapos (todos em uma única sala enviando e recebendo mensagens de todos os participantes ou em modo privado um-a-um). Utilize o GAE para hospedar o servidor desta aplicação.

Informação adicional: *no GAE as requisições são limitadas a um timeout de 60s, logo adote um tempo limite de 45s no long polling.*

14/15 - Questão

Com base no projeto da questão anterior e adotando WebSocket como protocolo de comunicação, reimplemente o chat usando o Heroku como servidor.

15/15 - Questão

Construa um sistema distribuído em Java que realize a simulação de uma transferência de dinheiro entre contas. As contas, conforme a figura apresentada abaixo, ficam em locais diferentes. As contas A, B e C possuem, respectivamente, saldos de R$200, R$980 e R$300. As operações que devem ser realizadas com garantia de consistência são:

- Transferir de A para B: R$ 100
- Transferir de C para B: R$ 500

- Transferir de C para A: R$ 300
- Transferir de B para A: R$ 1000

Informe qual o resultado final de cada conta.

Disposição geográfica dos componentes

Op - Operator
Tx - TxCoordenator
AM - AccountManager
A - AccountA
B - AccountB
C - AccountC

Observações:

1) adote a técnica de Two-Phase Commit e

2) todos os projetos devem ficar no Heroku, exceto o cliente que deve ser standalone.

Bom trabalho para todos!

Email para envio do link do github: aristofânio@hotmail.com