Atividades

Atividades sobre RMI, Tolerância a Falha, Interoperabilidade, Transações Distribuídas e Protocolos

Pontuação da Atividade Individual: 35 pontos

18/03/2017

Controle de Versão

Versão	Data	Descrição	Responsável
1.0	18/03/2017	Criação do documento (adicionado as questões de 1 a 5)	Ari

Informações para resolução das questões:

- Todos os códigos devem ser enviados para um repositório no GitHub e dentro dele deverá haver todas as pastas nomeadas com no formato "questao_xx", onde x é o número da questão em dois dígitos, exemplo: *questao_09*;
- Cada pasta deve conter todos os projetos para a solução da questão;
- Deve-se utilizar apenas codificações em Java e em outra linguagem, quando assim for expressamente solicitado;
- Todos as classes e métodos devem ser comentados de forma que fique explicado quais os papéis e responsabilidades dos participantes (classes), os estados ou as configurações (atributos) e os comportamentos (métodos);
- Todos os projetos devem ser do tipo Maven e ser executados em Docker e
- Considere cada nó como sendo um container docker, inclusive os repositórios.
- **Prazo de entrega: 14/04/2017**

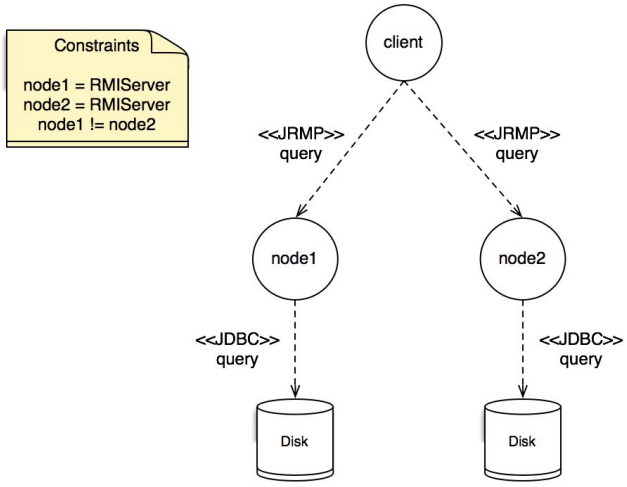
Segunda Linguagem (definido em 09/02/2017):

- Python - Zé
- C# - Julierme
- Scala - Aluísio
- DART - Pedro
- C++ - Isak

- Go - Michelle
- NodeJS - Wensttay
- VB - Laerton
- PHP - Asheley
- Pascal (OO) - Edilva
- Erlang - Zilderlan
- Rust - Dijalma

1/5 - Questão(2 pontos)

Considere o cenário onde uma aplicação cliente precisa se comunicar com dois bancos de dados para resolver uma única consulta. Desenvolva uma aplicação baseada em RMI/JRMP onde este cenário pode ser observado e descreva, pelo menos, quatro problemas diferentes que deveriam ser tratados para garantir a consistência dos dados. Utilize a topologia abaixo para responder esta questão.



Informações adicionais:
[node 0] aplicação cliente
[node 1] banco de dados postgresql
[node 2] banco de dados mysql

2/5 - Questão(4 pontos)

Considere o cenário descrito abaixo e implemente-o na sua segunda linguagem.

Em um parque existem N passageiros e dois carro em uma montanha russa. Os passageiros, repetidamente, esperam em duas filas, uma para cada carro, para dar uma volta na montanha russa. Cada carro tem capacidade para C passageiros.

Cada 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. Sempre escolhem a fila de menor tamanho ou, caso possuam o mesmo

tamanho, escolhem uma fila diferente da que escolheu na volta anterior. Cada volta na montanha russa leva 10ft (fake time = 1s). Cada passeio no parque de diversões leva entre 10ft e 30ft (aleatoriamente definido). Após 160ft o parque fecha.

Considere: C, N ∈ $\mathbb{Z}^+$; C > 20; C ≈ αN e α = 0.475.

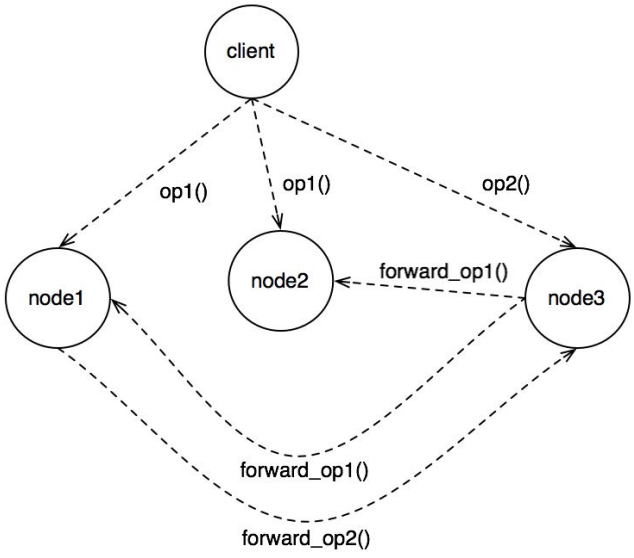
Dicas:

- utilize mensagens no console para rastrear tudo que está acontencendo;
- tanto os carros como os passageiros devem ser representados por threads ou equivalentes da sua linguagem e
- tanto os carros quanto os passageiros 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
```

3/5 - Questão(6 pontos)

Considere o cenário onde uma aplicação cliente possui conhecimento de onde encontram-se outros três nós, sendo que dois deles são iguais (réplicas) e que ao necessitar realizar uma consulta para qualquer um dos nós, caso não consiga, tentará em um outro **diferente**. Implemente este cenário o esquema ao lado. Utilize sua segunda linguagem para implementar esta questão.



Notas:

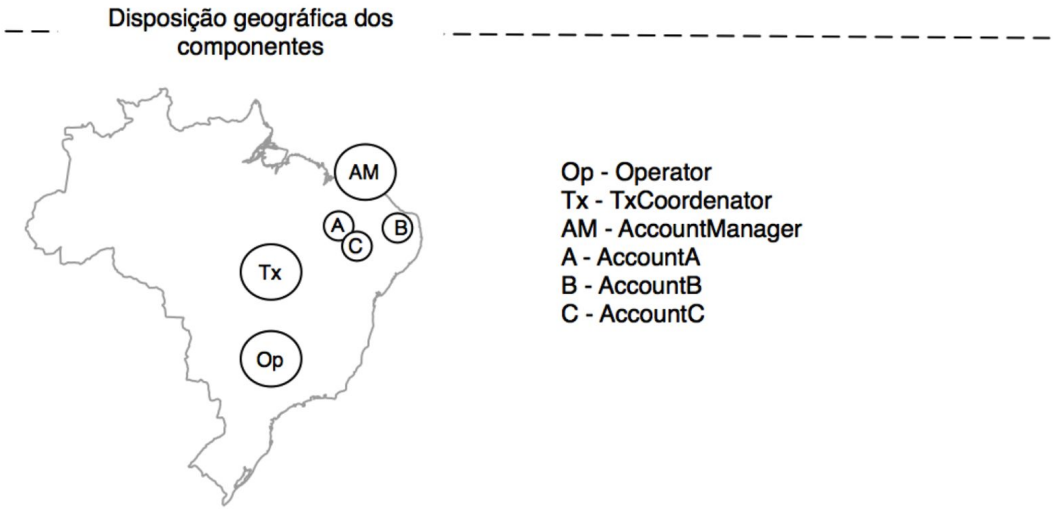
- siga a topologia ao lado;
- adote socket
- considere op1 = sum(x,y)
- considere op2 = diff(x,y)

Observação

- Node1 = Node2;
- Node1 e Node2 sabe resolver apenas soma;
- Node3 sabe resolver apenas diferença;
- Node1 deve delegar para Node3 a operação de diferença;
- Node3 deve delegar para Node1 a operação de soma e
- utilize socket para comunicação entre os nós

4/5 - Gerenciando Transações Distribuídas (8 pontos)

Um banco resolveu distribuir seu sistema de gerenciamento de contas em diversos locais diferentes para garantir maior disponibilidade para seus clientes. A cada minuto o sistema analisa qual a melhor rota entre seus servidores e cria uma disposição geográfica dos componentes e disponibiliza esta configuração para que o sistema funcione. Uma das disposições é apresentada na figura abaixo e deve ser nela que o sistema deverá operar no próximo minuto.



Para simular tal cenário, considerando esta disposição, construa um sistema de transação distribuída utilizando commit em duas fases para garantir que as operações abaixo serão ou não efetivadas, de acordo com a existência de saldos em contas, dentro de um tempo específico (5 segundos usando docker na sua máquina para simular localizações diferentes).

Operação	**Valor**	**Exec_X**	**Exec_Y**
Depósito na conta A	500.00	1	1

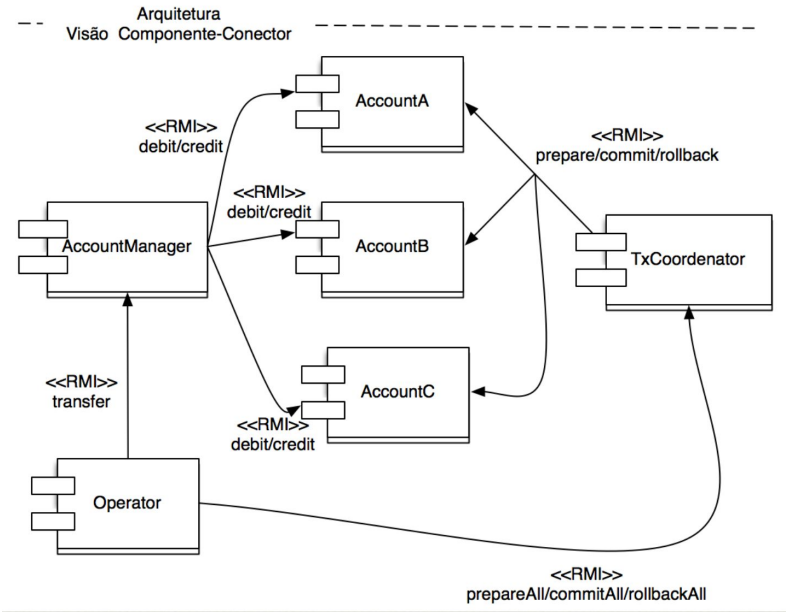
Depósito na conta B	1000.00	2	2
Depósito na conta C	900.00	3	3
Pagamento de taxa na conta A	10.25	4	14
Depósito na conta C	100.00	5	10
Transferência da conta B para conta A	125.00	6	11
Transferência da conta B para conta C	200.00	7	9
Pagamento de Taxa de Transferência pela conta B	12.00	8	8
Debito automático na conta A	100.00	9	7
Transferência da conta C para conta A	1000.00	10	4
Débito automática na conta C	300.00	11	5
Resgate de aplicação para a conta C	1000.00	12	6
Pagamento de taxa na conta B	12.50	13	13
Pagamento de taxa na conta C	32.25	14	12

Observe que existem duas sequência de execução destas operações e você deverá executá-las separadamente para responder as questões abaixo.

a) Qual o valor final de cada conta?
b) O tempo foi suficiente para executar as operações nas sequências pré-definidas?

Especificações adicionais:

- Adote a seguinte arquitetura de componentes para produzir o seu sistema.

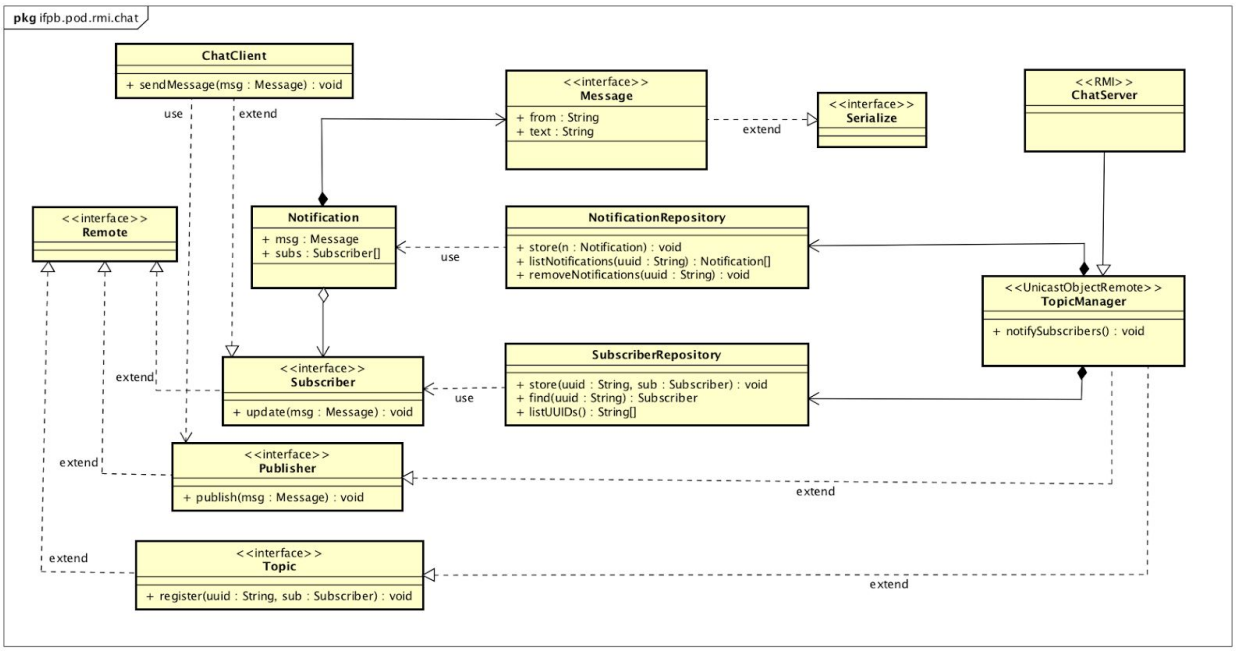


5/5 - Garantia de Entrega de Mensagem com RMI/JRMP (15 pontos)

Adotando o padrào Publisher/Subscriber, construa um chat que garanta a entrega da mensagem para todos os inscritos em um tópico chamado "chatgroup". Para tanto adote RMI/JRMP como teconologia para implementação das aplicações cliente-servidor.

Especificações:

- Utilize PostgreSQL como SGBD
- Utilize a arquitetura abaixo descrita
- Todo cliente deverá ser identificado pelo seu email
- A mensagem deverá ser armazenada por tempo ilimitado
- Os códigos de execução do cliente e do servidor derão seguir conforme indicado abaixo



Informações sobre a modelagem dos elementos da arquitetura:

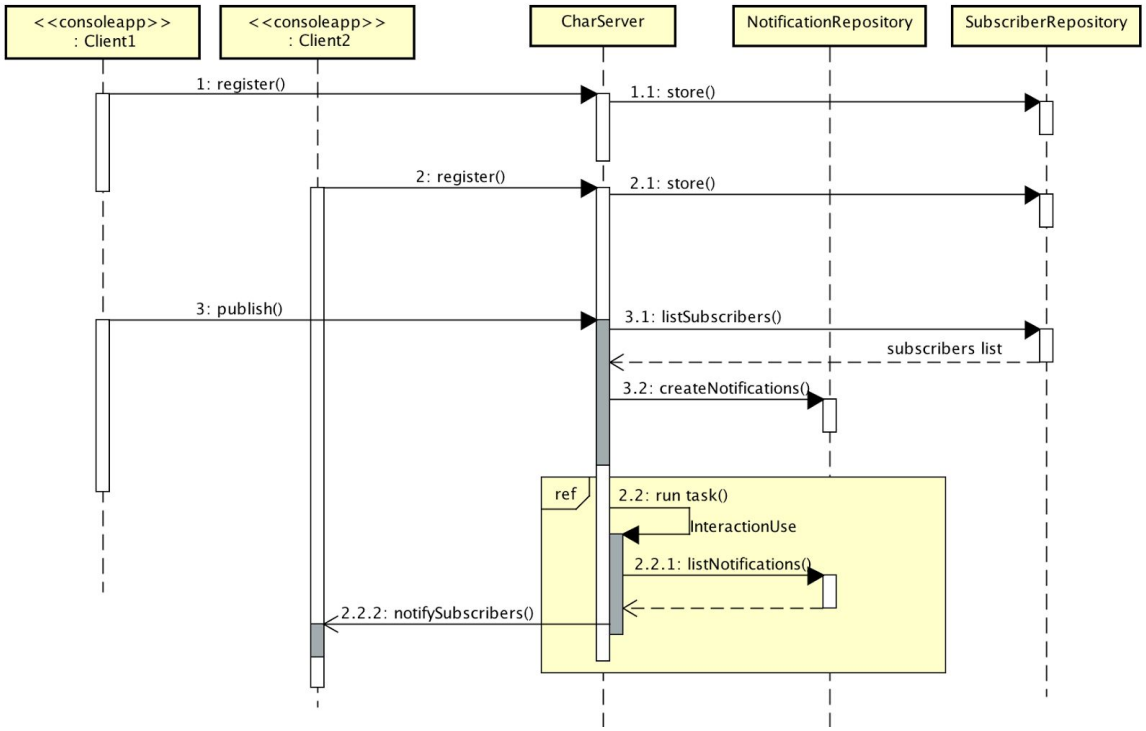
- Tanto **NotificationRepository** quanto **SubscriberRepository** deverão fazer uso de JDBC para amazenar e recuperar dados.
- **TopicManager** pode ser uma classe abstrata que é implementada por ChatServer, o qual deverá expor as interfaces as quais implementa **Publisher** e **Topic**.
- A notificação de um cliente (Subscriber) deverá ser feita a partir de uma rotina de atualização do lado do servidor, conforme exposto no código abaixo:

```
public class App {

  public static void main(String[] args) throws RemoteException, AlreadyBoundException {
    //
    final TopicManager manager = new TopicManagerImpl();
    //
    Registry registry = LocateRegistry.createRegistry(10999);
    registry.bind("__ChatServer__", manager);
    //
    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
      @Override
      public void run() {
        try {
          manager.notifySubscribers();
        } catch (RemoteException e) {
          e.printStackTrace();
        }
      }
    }, 1000, 10000);//1s, 10s
  }

}
```

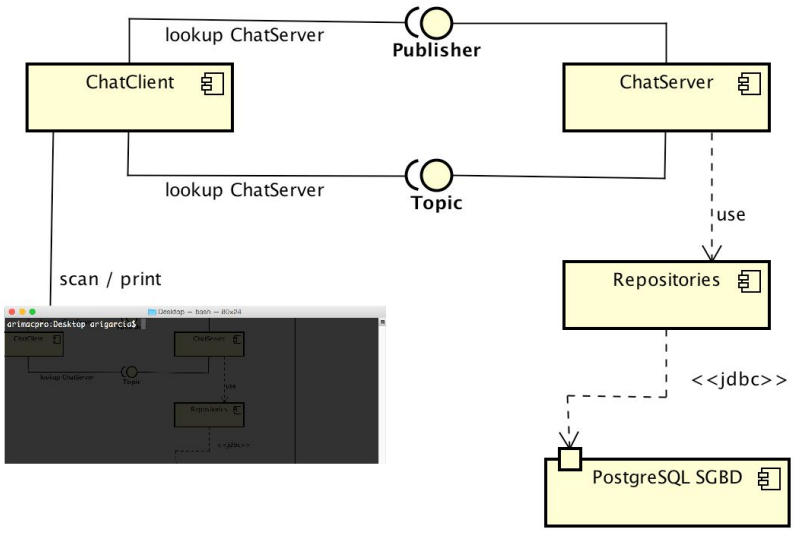
- O executor do lado do cliente deverá seguir o seguinte código:

```
public class App {

  public static void main(String[] args) throws RemoteException, NotBoundException {
    //
    String uuid = "<<seu email>>";
    //
    Registry registry = LocateRegistry.getRegistry(10999);
    Topic topic = (Topic) registry.lookup("__ChatServer__");
    Publisher publisher = (Publisher) registry.lookup("__ChatServer__");
    //
    ChatClientImpl client = new ChatClientImpl(publisher);
    topic.register(uuid, client);
    //
    Scanner scanner = new Scanner(System.in);
    while(true){
      //
      String text = scanner.nextLine();
      //
      Message message = new Message();
      message.setFrom(uuid);
      message.setText(text);
      //
      client.sendMessage(message);
    }

  }

}
```

- O fluxo de execução de um registro e um envio de mensagem para um cliente deverá ser o seguinte:



- Componentes da arquitetura:



5/5 - Gerenciando Transações Distribuídas (10 pontos)

Um banco resolveu distribuir seu sistema de gerenciamento de contas em diversos locais diferentes para garantir maior disponibilidade para seus clientes. A cada minuto o sistema analisa qual a melhor rota entre seus servidores e cria uma disposição geográfica dos componentes e disponibiliza esta configuração para que o sistema funcione. Uma das disposições é apresentada na figura abaixo e deve ser nela que o sistema deverá operar no próximo minuto.

Disposição geográfica dos componentes

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

Para simular tal cenário, considerando esta disposição, construa um sistema de transação distribuída utilizando commit em duas fases para garantir que as operações abaixo serão ou não efetivadas, de acordo com a existência de saldos em contas, dentro de um tempo específico (5 segundos usando docker na sua máquina para simular localizações diferentes).

Operação	**Valor**	**Exec_X**	**Exec_Y**
Depósito na conta A	500.00	1	1
Depósito na conta B	1000.00	2	2
Depósito na conta C	900.00	3	3
Pagamento de taxa na conta A	10.25	4	14
Depósito na conta C	100.00	5	10
Transferência da conta B para conta A	125.00	6	11
Transferência da conta B para conta C	200.00	7	9
Pagamento de Taxa de Transferência pela conta B	12.00	8	8
Debito automático na conta A	100.00	9	7
Transferência da conta C para conta A	1000.00	10	4
Débito automática na conta C	300.00	11	5
Resgate de aplicação para a conta C	1000.00	12	6
Pagamento de taxa na conta B	12.50	13	13

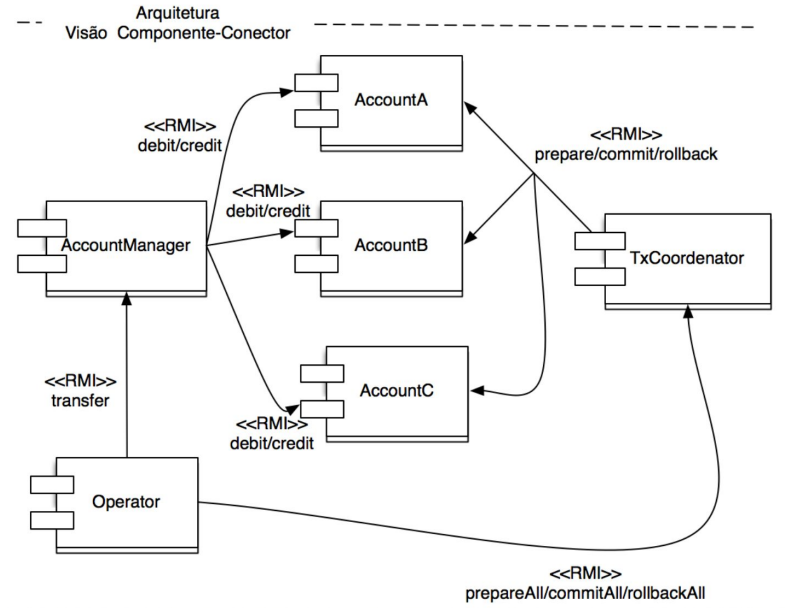
Pagamento de taxa na conta C	32.25	14	12

Observe que existem duas sequência de execução destas operações e você deverá executá-las separadamente para responder as questões abaixo.

c) Qual o valor final de cada conta?
d) O tempo foi suficiente para executar as operações nas sequências pré-definidas?

Especificações adicionais:

- Adote a seguinte arquitetura de componentes para produzir o seu sistema.



Bom trabalho para todos!

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