Atividade Colaborativa

Assuntos: Estilos Arquiteturais, Comunicação, Concorrência

Pontuação da Atividade Colaborativa: 50 pontos

22/09/2017

Controle de Versão

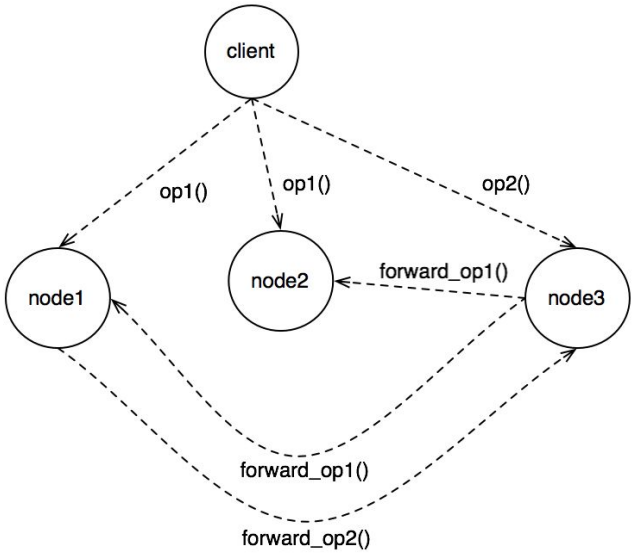
Versão	Data	Descrição	Responsável
1.0	01/02/2017	Criação do documento (adicionado as questões de 1 a 2)	Ari
2.0	28/02/2017	Adicionado as questões de 3 a 10	Ari
3.0	17/03/2017	Adicionado observação na questão 02	Ari, Laerton
4.0	18/03/2017	Adicionado observações nas questões 02 a 06	Ari, Wenstay
5.0	10/06/2017	Atualização para 2017.1	Ari
6.0	22/09/2017	Atualização para 2017.1 - alteração de questões para RMI	Ari
7.0	06/10/2017	Alterado a equação de entrada e saída da questão 03	Ari
8.0	13/03/2018	Alteração das informações para resolução das questões	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 resolução da questão correspondente;
- Deve-se utilizar apenas codificações em Java, exceto onde for expressamente solicitado outra linguagem;
- Todos as classes e métodos devem ser comentados de forma que fiquem explicados quais os papéis e responsabilidades dos participantes (classes), os estados ou as configurações (atributos) e os comportamentos (métodos);
- Adote Maven como arquetipo dos projetos
- Preparar slides detalhando como foi resolvido cada uma das questões
- **Entregar até o dia 06/04/2017**

1 - Questão **(estilo arquitetural baseado em objetos - 3 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 requisição para qualquer um dos nós, caso não consiga, tentará em um outro **diferente (não réplica)**. Implemente este cenário o esquema ao lado.



Notas:

- siga a topologia ao lado;
- adote RMI;
- retornar um RemoteException com a mensagem de erro ao realizar operação de divisão por zero.

Observação

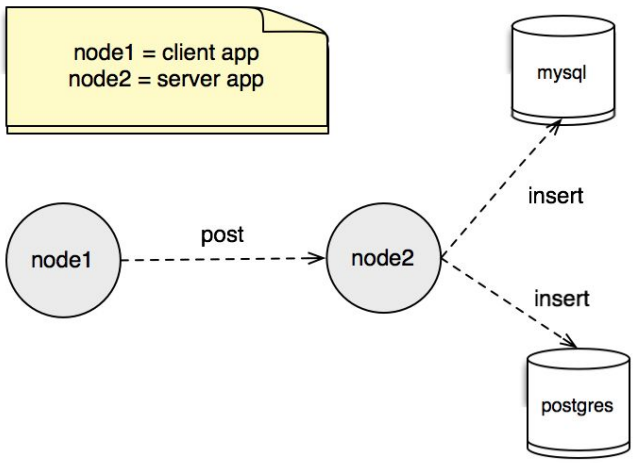
- Node2 é réplica de Node1;
- Node1 e Node2 sabe resolver apenas op1;
- Node3 sabe resolver apenas op2;
- Node1 deve delegar para Node3 a operação de op2;
- Node3 deve delegar para Node1 a operação de op1 e
- Considere a formulação para

$$op1(x,y) = 2yx$$

$$op2(x,y) = 2x/y$$

2 - Questão (estilo arquitetural baseado em camadas - 3 pontos)

Implemente um replicador de dados simples que garanta a consistência dos dados. Adote a topologia do sistema distribuído ao lado para isto e teste conforme instruções abaixo.
(Utilize RMI para comunicação entre os nós).



Informações complementares:

Tabela	Campos	Banco
customer	code int name varchar(50)	Mysql / Postgres

Testes:

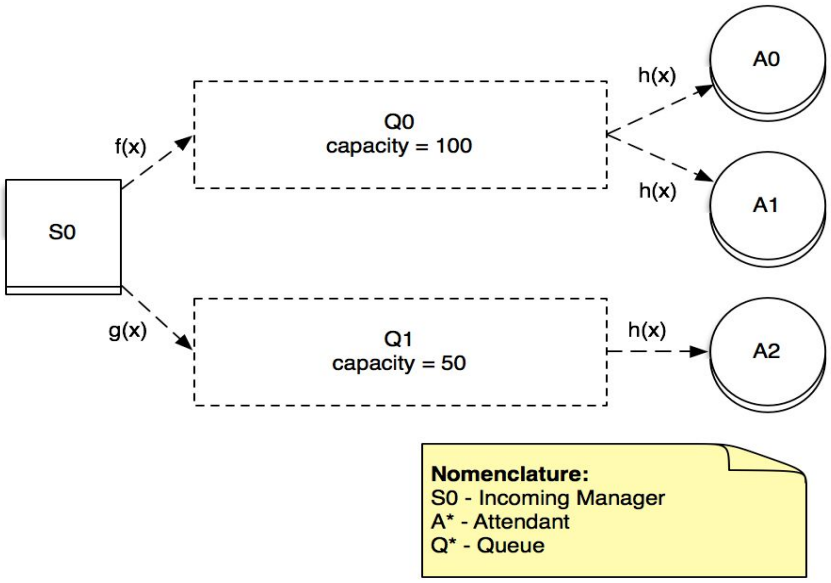
Carga	Procedimento	Resultado
100 usuários	Envie 100 requisições de inserção de dados de **customer** e informe o tempo total (em segundos)	
1000 usuários	Envie 1000 requisições de inserção de dados de **customer** e informe o tempo total (em segundos)	
1000 usuários	Adotando threads, implemente uma solução que reduza o tempo de inserção a partir de 1000 requisições e informe o tempo total (em segundos)	

3 - Questão (concorrência - 5 pontos)

Conforme ilustrado abaixo, o sistema de atendimento ocorre em duas filas distintas. A primeira fila (Q0) possui capacidade para 100 pessoas e a segunda (Q1) possui capacidade para 50 pessoas. A primeira fila é atendida por dois postos de trabalho, enquanto que a segunda

apenas por um. A taxa de chegada para a primeira fila é de f(x) pessoas por segundo e a taxa de chegada para a segunda fila é de g(x) pessoas por segundo. A taxa de atendimento é de h(x) pessoas por segundo em qualquer um dos postos de trabalho. Considere $x \in \Re$ obtido aleatoriamente a cada 1 segundo, sendo que $0 \leq x < 1.$ Considere ainda que: $f(x) = 0.833e^{-x}$, $g(x) = 2x$ e $h(x) = 0.3x^{x}$. Com tais informações construa uma aplicação que simule este sistema de atendimento e responda as seguintes perguntas no fim de 600 segundos:

a) Quantas pessoas entraram na fila?
b) Quantas pessoas foram embora antes de entrar na fila (ocorre quando a fila está cheia)?
c) Quantas pessoas foram atendidas?
d) Quantas pessoas ficaram na fila?



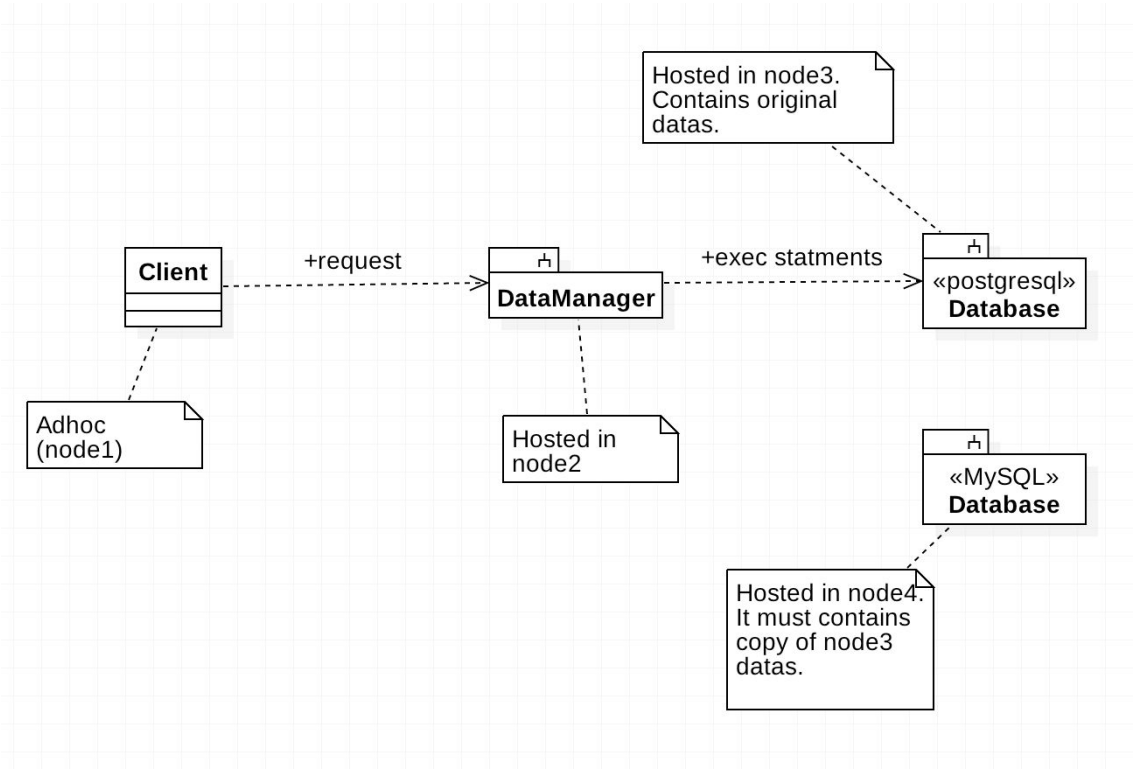
Informações adicionais:

- *na implementação, considere S0 como sendo uma máquina separada das demais implementações;*
- *use RMI para comunicação;*
- *deve-se adotar threads e os mecanismos de bloqueio e sincronização para desenvolver o sistema de simulação e*
- *use impressões no console para explicar a ocorrência dos eventos.*

4 - Questão **(replicação de dados - 5 pontos)**

Um cliente remoto realizar CRUD em um banco de dados através de um serviço disponibilizado na internet. Embora este sistema esteja funcionando perfeitamente, faz-se necessário agora a inclusão de um requisito adicional, a replicação de dados. O diagrama abaixo apresenta a disposição dos elementos da rede. Entendendo que o sistema precisa ser ampliado, desenvolva uma solução sem alterar a arquitetura atual para garantir que os dados serão replicados sob os seguintes limites:

- a aplicação cliente não pode ser alterada;
- qualquer alteração em DataManager deve ser baseado no envio de eventos para um barramento local ou remoto e
- sua solução poderá ter acesso direto ao banco de dados original, mas apenas no modo de somente-leitura.



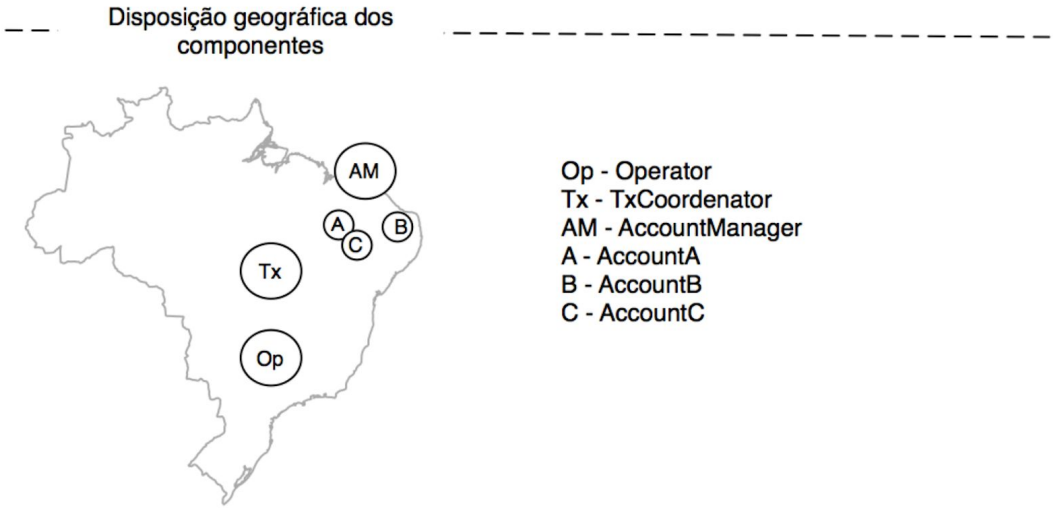
5 - Questão **(replicação de dados - 5 pontos)**

Implemente a questão anterior usando o MessageBus Patterns e informe qual o tempo médio de inconsistência.

- Observação: maior objetivo é replicação de dados, erros devem ser tratados apenas para este fim.

6 - Questão (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).

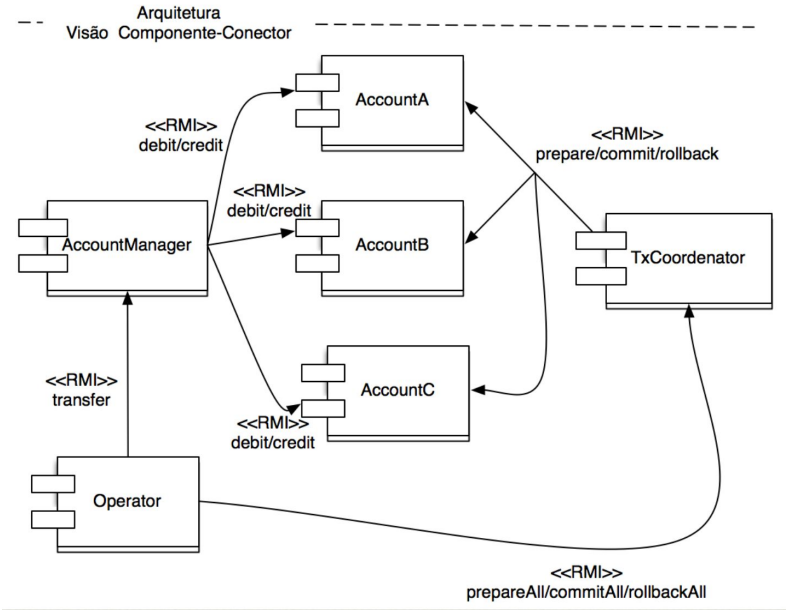
Operação	**Valor**	**Exec1**	**Exec2**
Depósito na conta A	500.00	1	1
Depósito na conta B	1000.00	2	2
Depósito na conta C	900.00	3	3
Pagamento de taxa na conta A	10.25	4	14
Depósito na conta C	100.00	5	10
Transferência da conta B para conta A	125.00	6	11
Transferência da conta B para conta C	200.00	7	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 às questões abaixo:

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

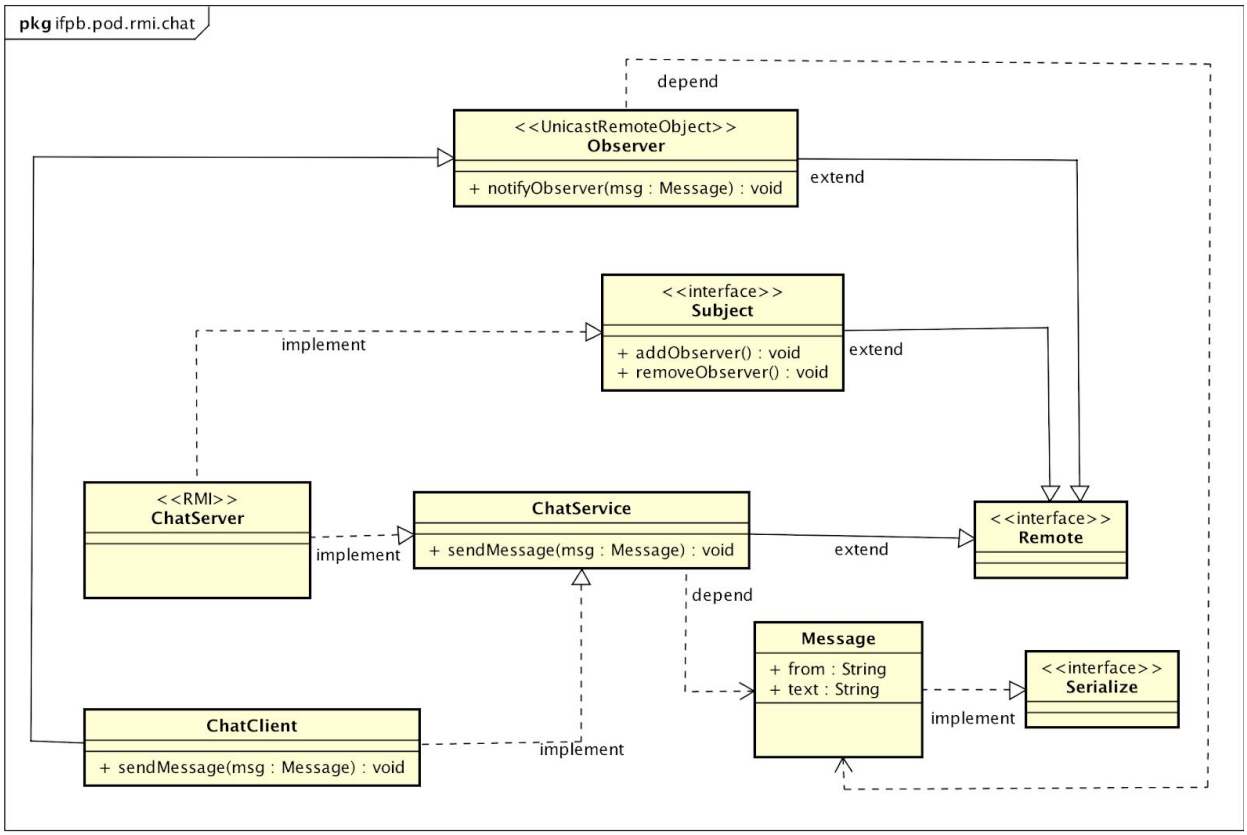
Especificações adicionais:
- Adote a seguinte arquitetura de componentes para produzir o seu sistema.



7 - Questão **(fazendo um Observer com RMI/JRMP - 8 pontos)**

Desenvolva um chat utilizando o padrão observer com RMI. Este chat deverá seguir as especificações abaixo descritas.

- **Nome do pacote base**: ifpb.pod.rmi.chat
- **Quantidade de Projetos:** 3 (um para o servidor, um para o cliente e outro para os códigos compartilhados)



O código de execução do cliente deverá ser:

```
public class App {

  public static void main(String[] args) throws RemoteException, NotBoundException {
    //
    String uuid = UUID.randomUUID().toString();
    //
    Registry registry = LocateRegistry.getRegistry(10999);
    Subject subject = (Subject) registry.lookup("__ChatServer__");
    ChatService service = (ChatService) registry.lookup("__ChatServer__");
    //
    ChatClient client = new ChatClient(service);
    subject.addObserver(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 código de execução do servidor deverá ser:

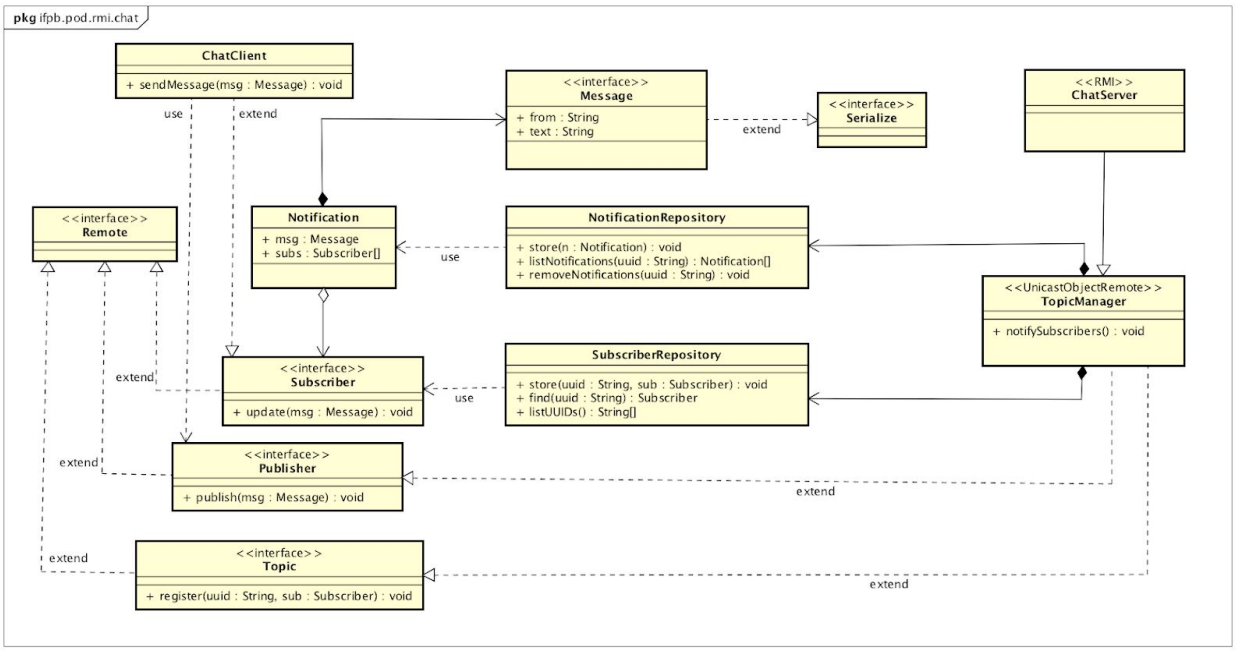
```
public class App {

  public static void main(String[] args) throws RemoteException, AlreadyBoundException {
    Registry registry = LocateRegistry.createRegistry(10999);
    registry.bind("__ChatServer__", new ChatServer());
  }

}
```

8 - Questão **(Garantia de Entrega de Mensagem com RMI/JRMP - 13 pontos)**

Adotando o padrão 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 usuário deverá ser identificado pelo seu email
- A mensagem deverá ser armazenada por tempo ilimitado
- Os códigos de execução do cliente e do servidor deverão seguir conforme indicado abaixo:



**nota 05-10-2017: Message não é interface, mas sim classe abstrata.*

Informações sobre a modelagem dos elementos da arquitetura:

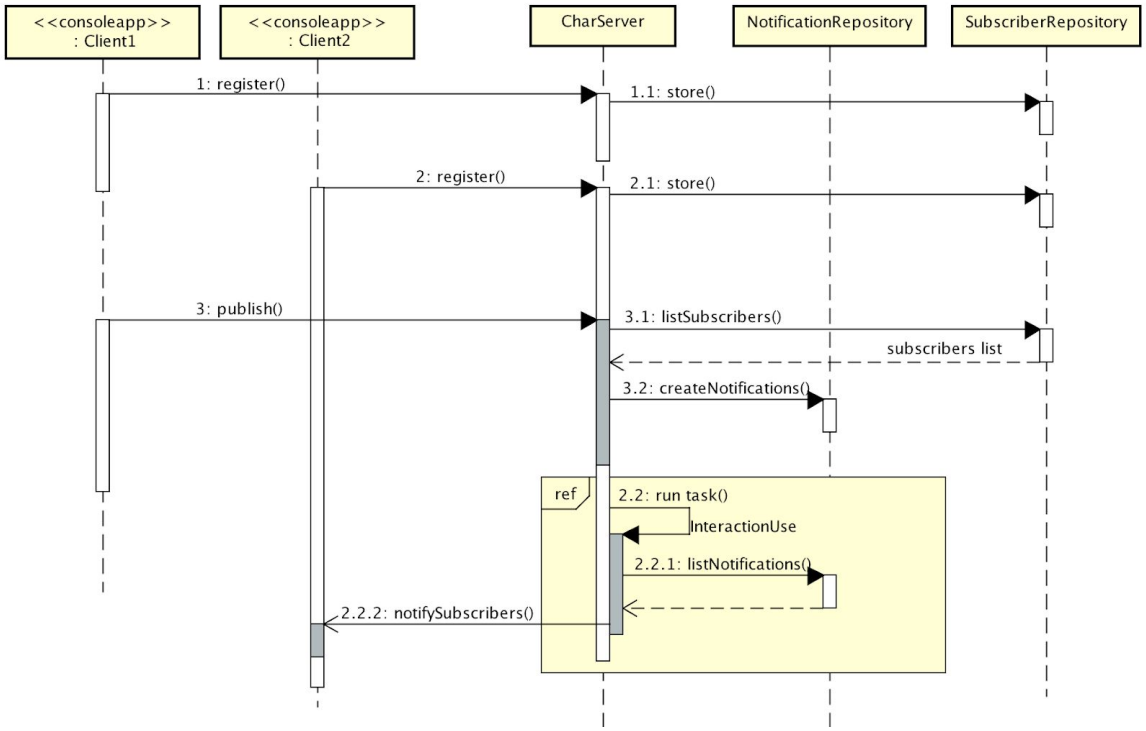
- Tanto **NotificationRepository** quanto **SubscriberRepository** deverão fazer uso de JDBC para amazenar e recuperar dados.
- **TopicManager** pode ser uma classe abstrata que é implementada por ChatServer, o qual deverá expor as interfaces as quais implementa **Publisher** e **Topic**.
- A notificação 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:

Bom trabalho para todos!

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