DIEGO SOUSA SANTOS – 11044616



RELATÓRIO EP-KVSTORE – SISTEMAS DISTRIBUÍDOS

SÃO CAETANO DO SUL – SP

11/2022

LINK DO VÍDEO:

1. FORMATO DA MENSAGEM TRANSFERIDA

As mensagens são transferidas através do objeto Message, descrito no diagrama UML abaixo:



Figura UML da classe Message

O atributo “id” define o identificador da mensagem através de um UUID que permite tornar única cada mensagem.

O atributo “type” é um struct que classifica a mensagem de acordo com os tipos PUT, GET, REPLICATION, GET\_RESPONSE, PUT\_OK, PUT\_NOK, TRY\_OTHER\_SERVER\_OR\_LATER, REPLICATION\_OK, REPLICATION\_NOK, possibilitando diferenciar esses tipos de mensagens.

O atributo “key” é o identificador do objeto referenciado na mensagem.

O atributo “value” é o valor do objeto referenciado na mensagem.

O atributo “timeStamp” é a referência temporal do cadastro do objeto referenciado na mensagem.

O atributo “clientPort” é a porta de origem da mensagem.

O método “initializeUUID” gera o UUID da mensagem.

O método “toJson” utiliza a biblioteca gson para converter o objeto Mensagem em Json.

Os métodos “setAs...” classificam e criam a mensagem de acordo com os parâmetros exigidos para cada tipo especificado.

O método estático “timeStampToString” retorna uma String a partir de um LocalDateTime.

O método estático “stringToDateTime” retorna um LocalDateTime a partir de uma String.

O método estático “fromJson” retorna uma Mensagem a partir de um json.

1. CLIENT

2.1 MÉTODO MAIN

No método main existe um loop para leitura dos dados do usuário. É impresso um menu composto das opções “1. INIT”, “2. PUT” e “3. GET”, as quais o usuário pode selecionar informando o número ou o texto da opção.

Caso a opção de INIT seja selecionada, o usuário deverá informar as três portas dos servidores envolvidos no sistema distribuído. Após isso, cada porta especificada é inserida no Set “serverPorts” do cliente, para armazenar os servidos conhecidos.

Caso a opção de PUT seja selecionada, o usuário deverá informar primeiro a chave e, logo após, o valor do objeto key-value a ser registrado nos servidores.

Caso a opção de GET seja selecionada, o usuário deverá informar apenas a chave do objeto procurado.

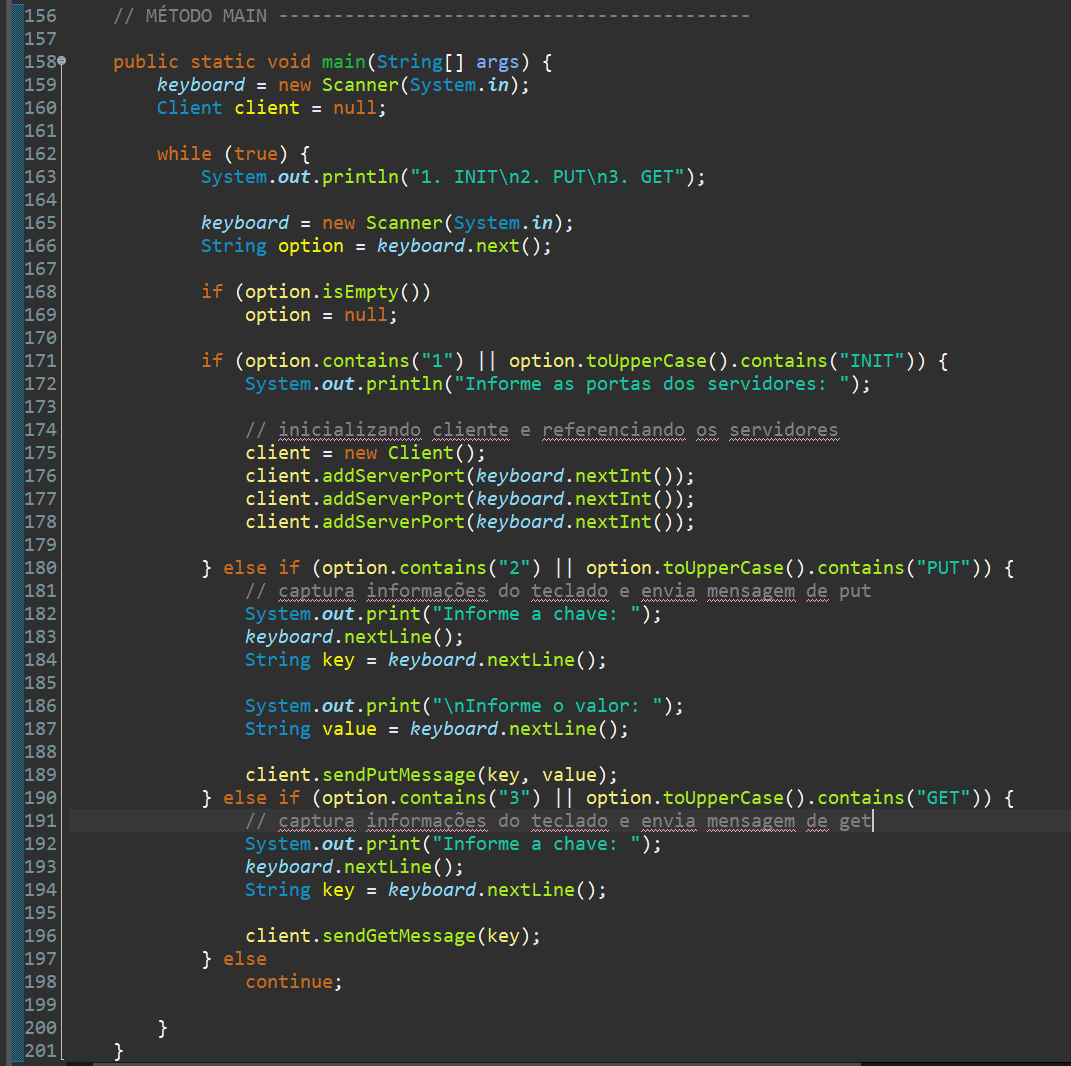


Figura método main da classe Client

2.2 DEFINIÇÃO DO CLIENT

Os peers são definidos através da classe descrita no diagrama UML abaixo:

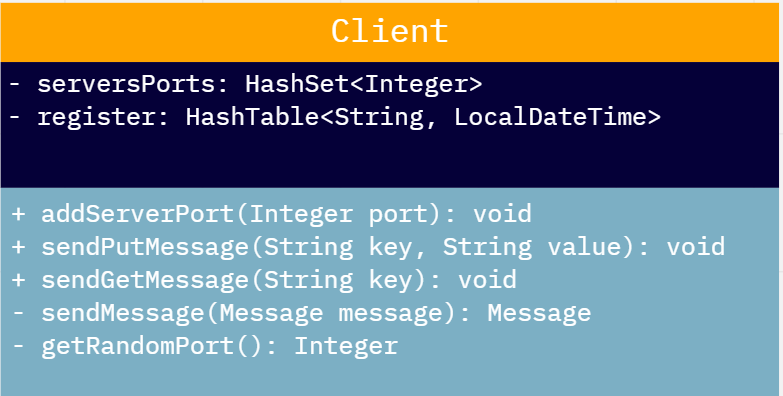


Figura UML da classe Client

2.2.1 ATRIBUTOS

O atributo “serversPorts” conjunto de portas únicas de servidores conhecidos, informados no método main.

O atributo “register” armazena o registro que foram cadastrados com sucesso nos servidores através de mensagens PUT.

2.2.2 MÉTODOS

**addServerPort**:

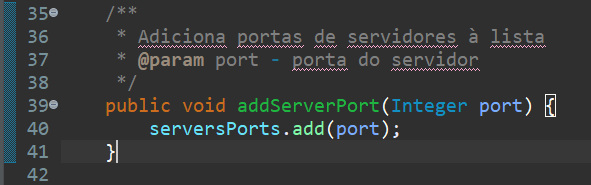


Figura método addServerPort

**sendPutMessage:** Envia de forma assíncrona uma mensagem do tipo PUT com os parâmetros informados pelo usuário. Se a resposta da requisição for um PUT\_OK, o registro é salvo no register do client.

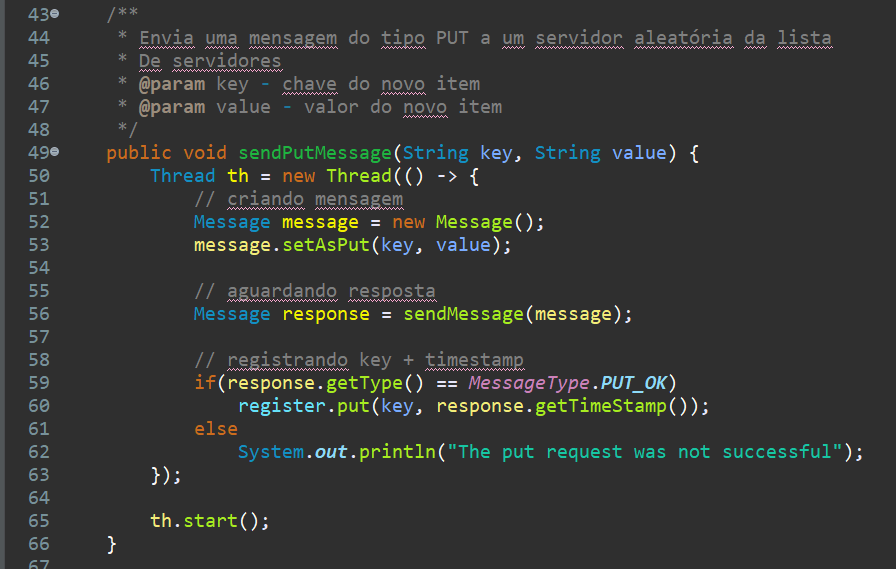


Figura método sendPutMessage

**sendGetMessage:** Envia de forma assíncrona uma mensagem do tipo GET com os parâmetros informados pelo usuário.



Figura método sendGetMessage

**sendMessage:** Envia a mensagem genérica (PUT ou GET) para um dos servidores registrados aleatoriamente. Após isso, aguarda a resposta e imprime o resultado na tela.



Figura método sendMessage

**getRandomPort:** retorna uma porta aleatória dentre as que estão registradas na lista serverPorts.

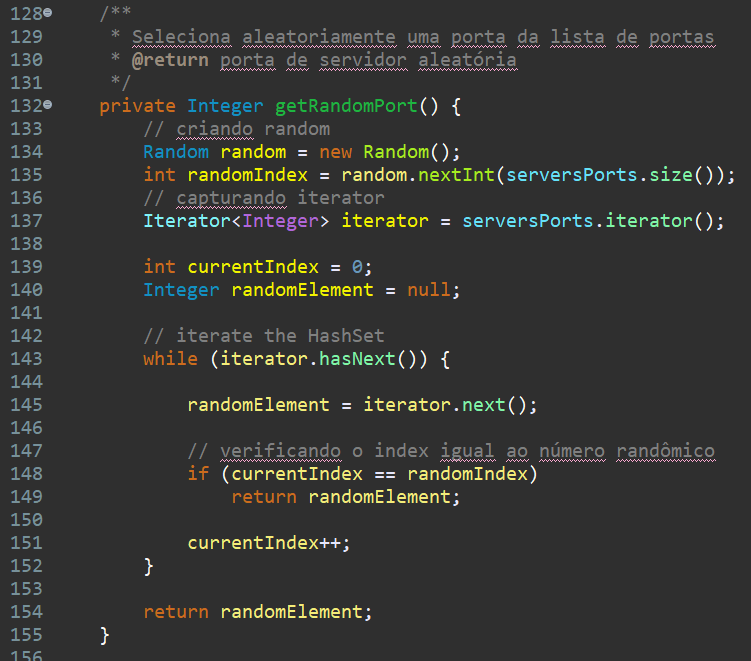


Figura método getRandomPort

1. SERVER

3.1 MÉTODO MAIN

No método main são capturados na sequência: IP do servidor, porta do servidor e porta do líder. Após isso, a instância do servidor é inicializada.



Figura método main da classe Server

3.2 DEFINIÇÃO DO SERVER

Os peers são definidos através da classe descrita no diagrama UML abaixo:

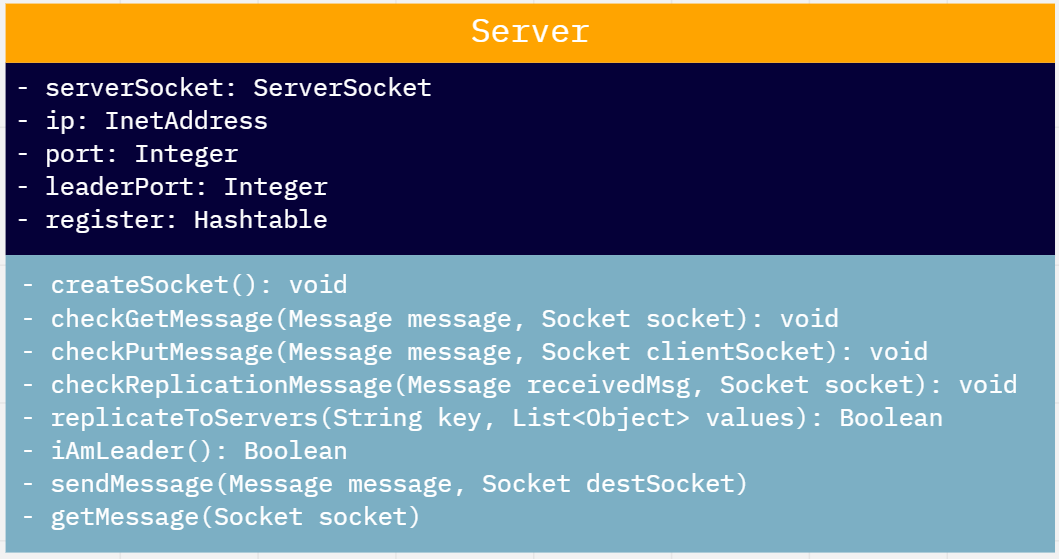


Figura UML da classe Server

3.2.1 ATRIBUTOS

O atributo “serverSocket” referencia a instância de ServerSocket criada juntamente com o objeto Server.

O atributo “ip” refere-se ao ip do servidor.

O atributo “port” refere-se à porta do servidor.

O atributo “leaderPort” registra a porta do servidor líder do sistema distribuído.

O atributo “register” é uma hashtable que armazena os objetos key-value que são registrados no servidores através das mensagens do cliente.

3.2.2 MÉTODOS

**createSocket**: responsável por inicializar o socket da instância de Server em questão. O socket é inicializado de forma assíncrona, e após isso é criado um loop para identificar e tratar o recebimento de mensagens.



Figura método createSocket

**getMessage:** responsável por, de forma assíncrona, capturar o pacote recebido com a mensagem do client/server e, a partir do tipo da mensagem direcionar o tratamento correto.

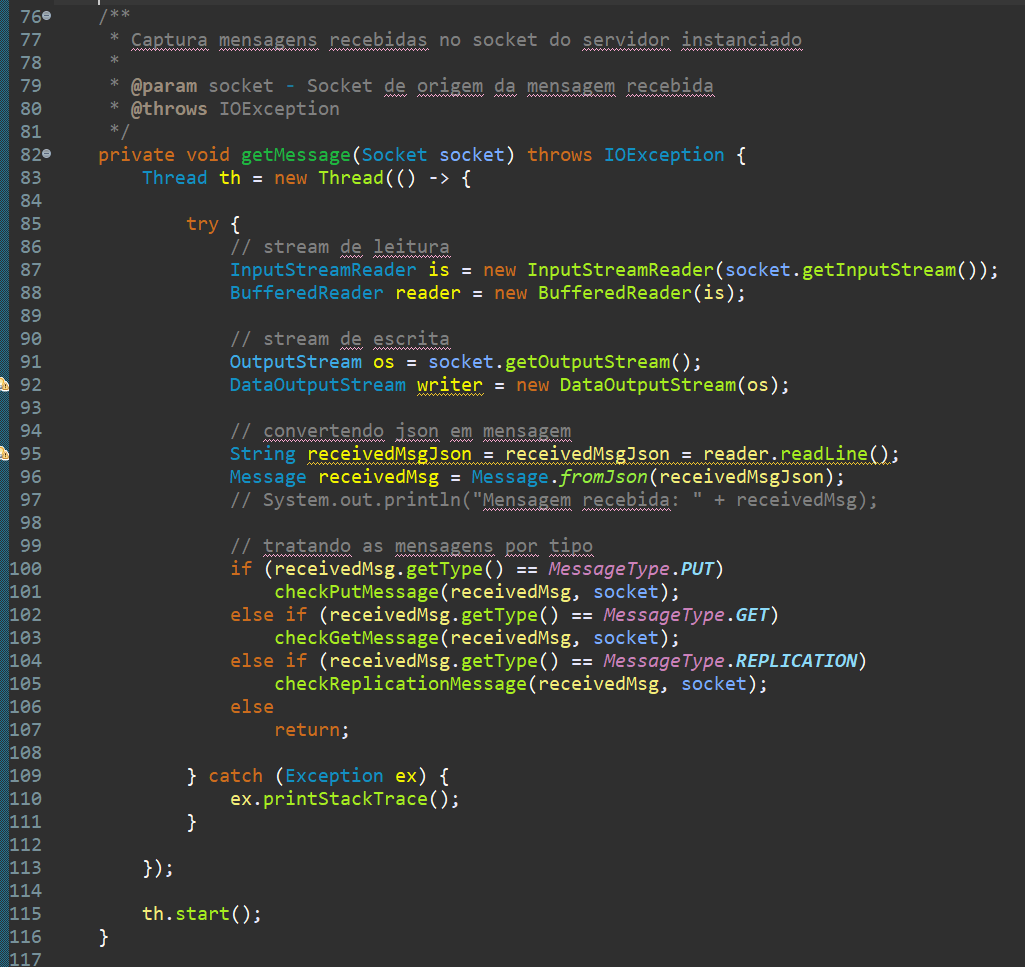


Figura método getMessage

**checkGetMessage**: responsável por tratar as mensagens recebidas do tipo GET. Com a chave informada na mensagem, o servidor procura o objeto no register e, caso não exista, retorna nulo. Caso exista, o servidor retorna o objeto procurado. **OBSERVAÇÃO: para simular um pacote desatualizado foi criada uma variável random (linhas 137 a 140) para gerar falhas aleatórias, com 30% de chances de ocorrer. Isso porque como o sistema roda em local host, a chance de ocorrerem falhas na replicação é praticamente nula.**

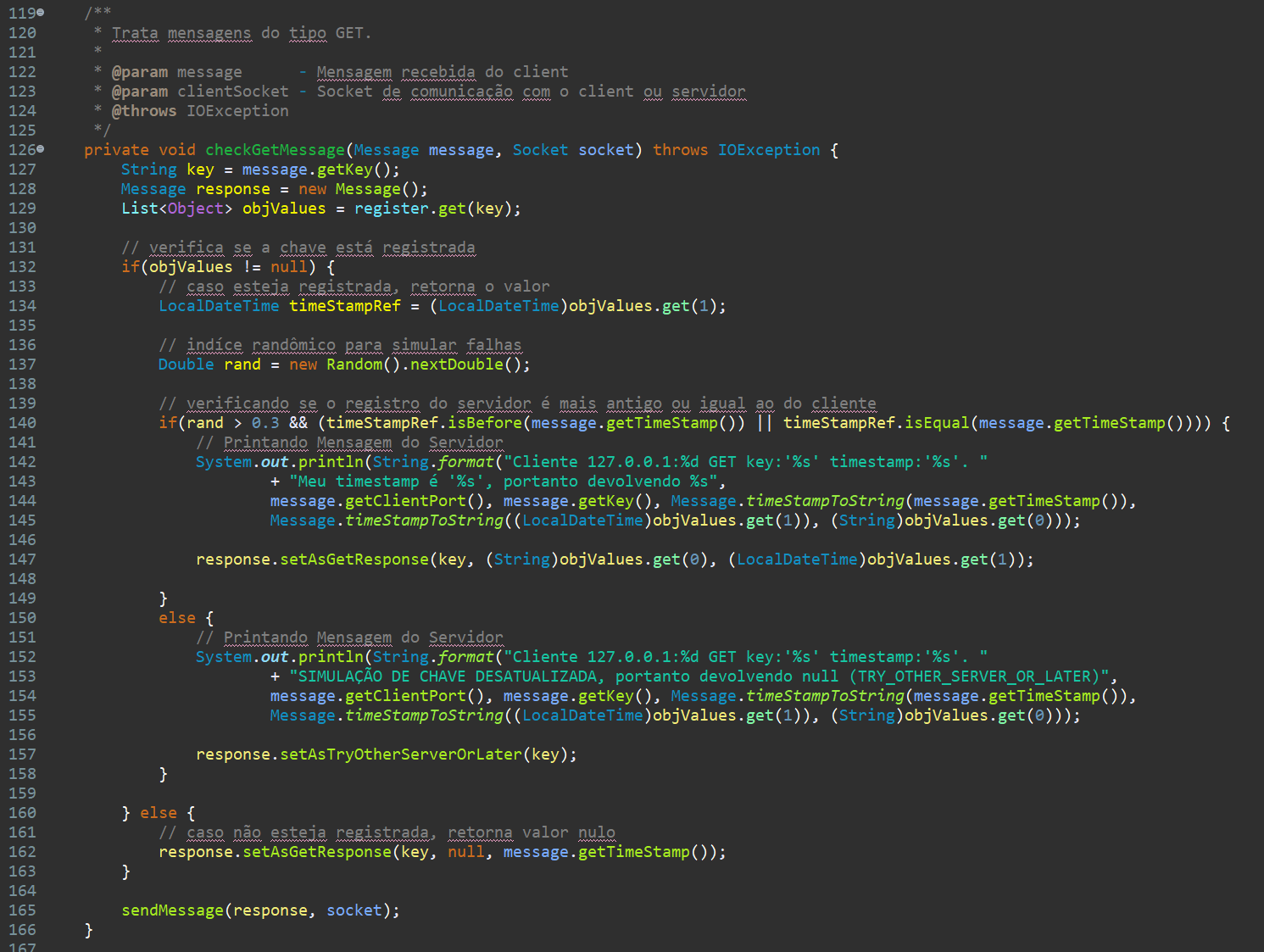


Figura método checkGetMessage

**checkPutMessage**: responsável por tratar as mensagens do tipo PUT. A partir dos parâmetros recebidos, se o servidor for líder o objeto é cadastrado no register. Caso contrário, a mensagem é redirecionada ao líder. Após isso, o líder replica o objeto para os demais servidores e, ocorrendo a replicação com sucesso, a resposta é enviada ao cliente como PUT\_OK, com o timestamp do registro. Caso contrário, é enviado um PUT\_NOK confirmando a falha no registro.

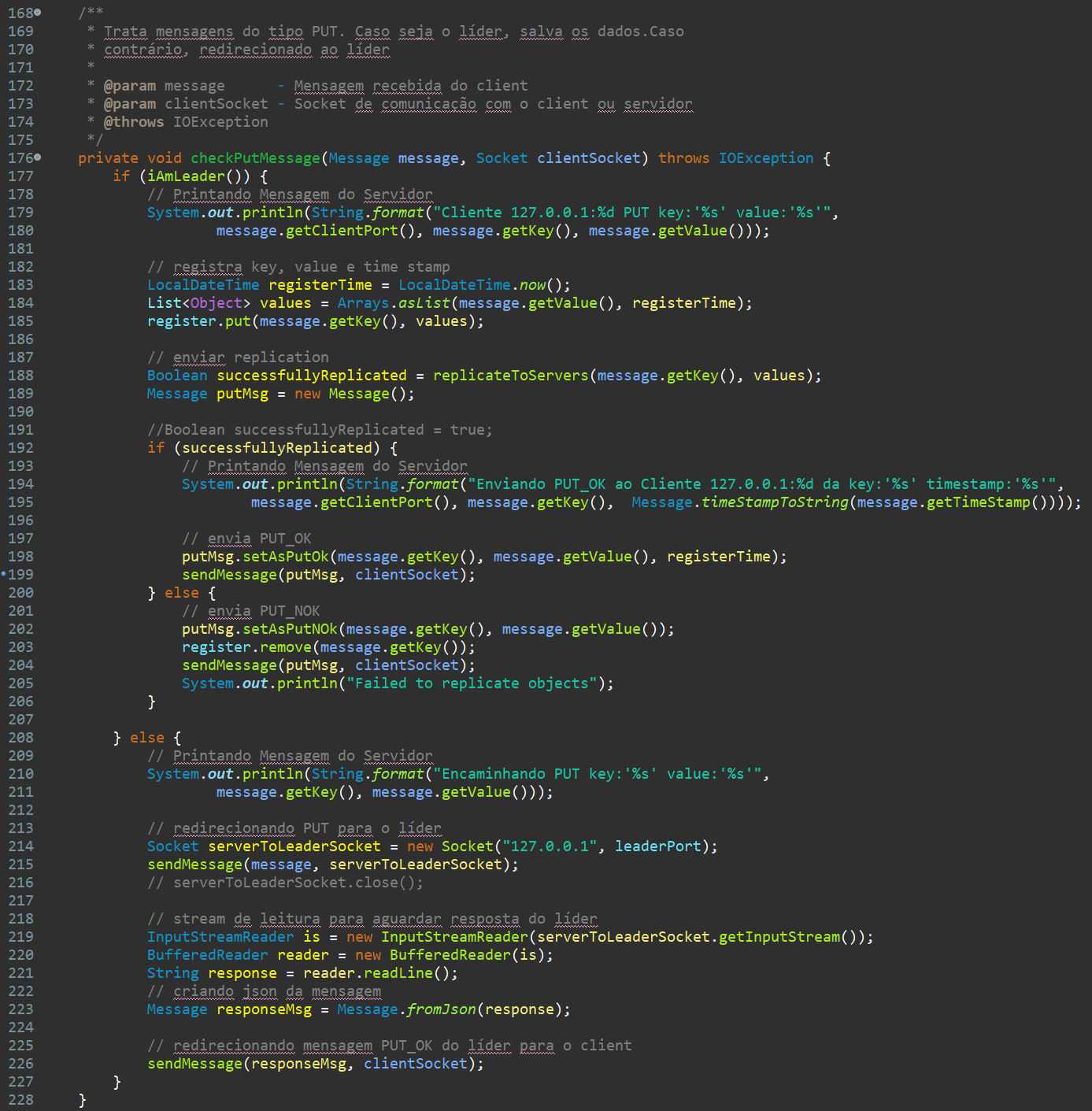


Figura método checkPutMessage

**checkReplicationMessage**: responsável por tratar mensagens do tipo Replication. Com base nos parâmetros recebidos na mensagem, o objeto é cadastrado no register do servidor caso ele não seja o líder e devolve uma mensagem REPLICATION\_OK. Caso contrário, devolve uma mensagem REPLICATION\_NOK.

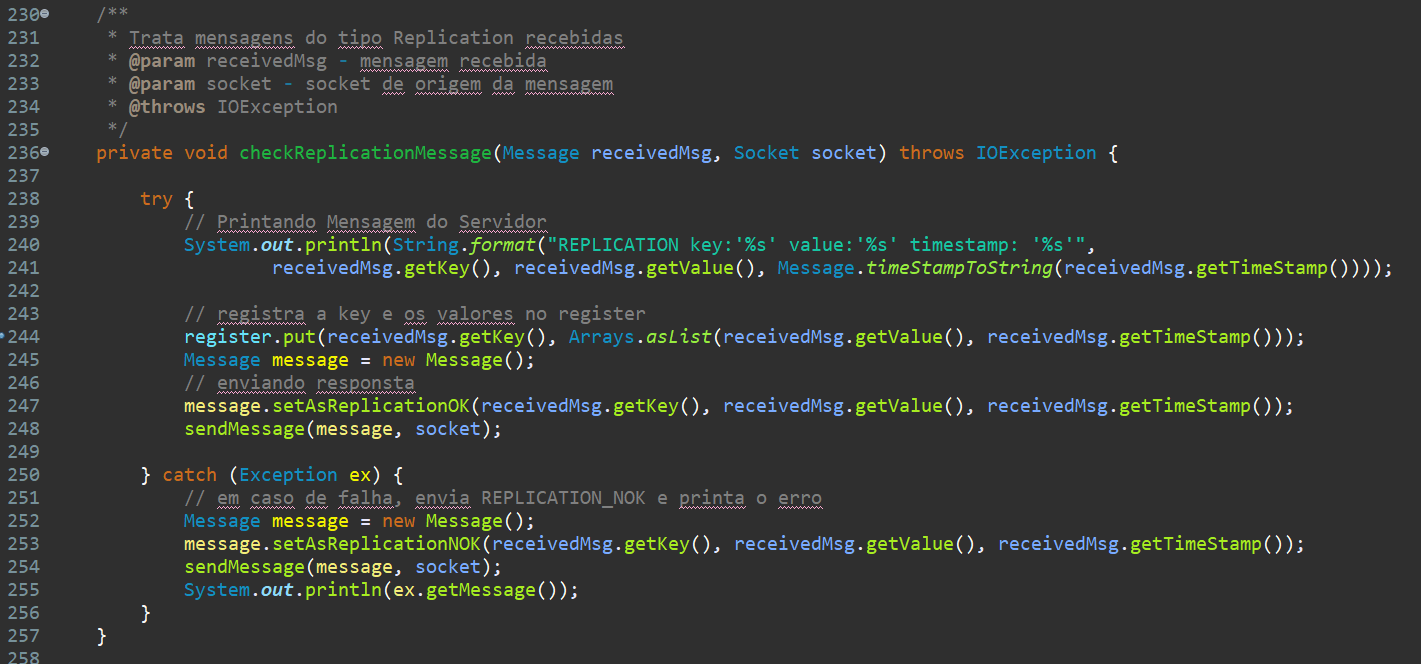


Figura método checkReplicationMessage

**replicateToServers:** responsável por replicar, a partir do líder, uma mensagem aos demais servidores. A lista de portas de servidores está fixada em 10097, 10098 e 10099. Para cada porta, o método mapeia um socket para enviar uma mensagem do tipo REPLICATION para cadastro dos objetos. Após isso o líder aguarda a resposta e o método devolve true ou false de acordo com ela.

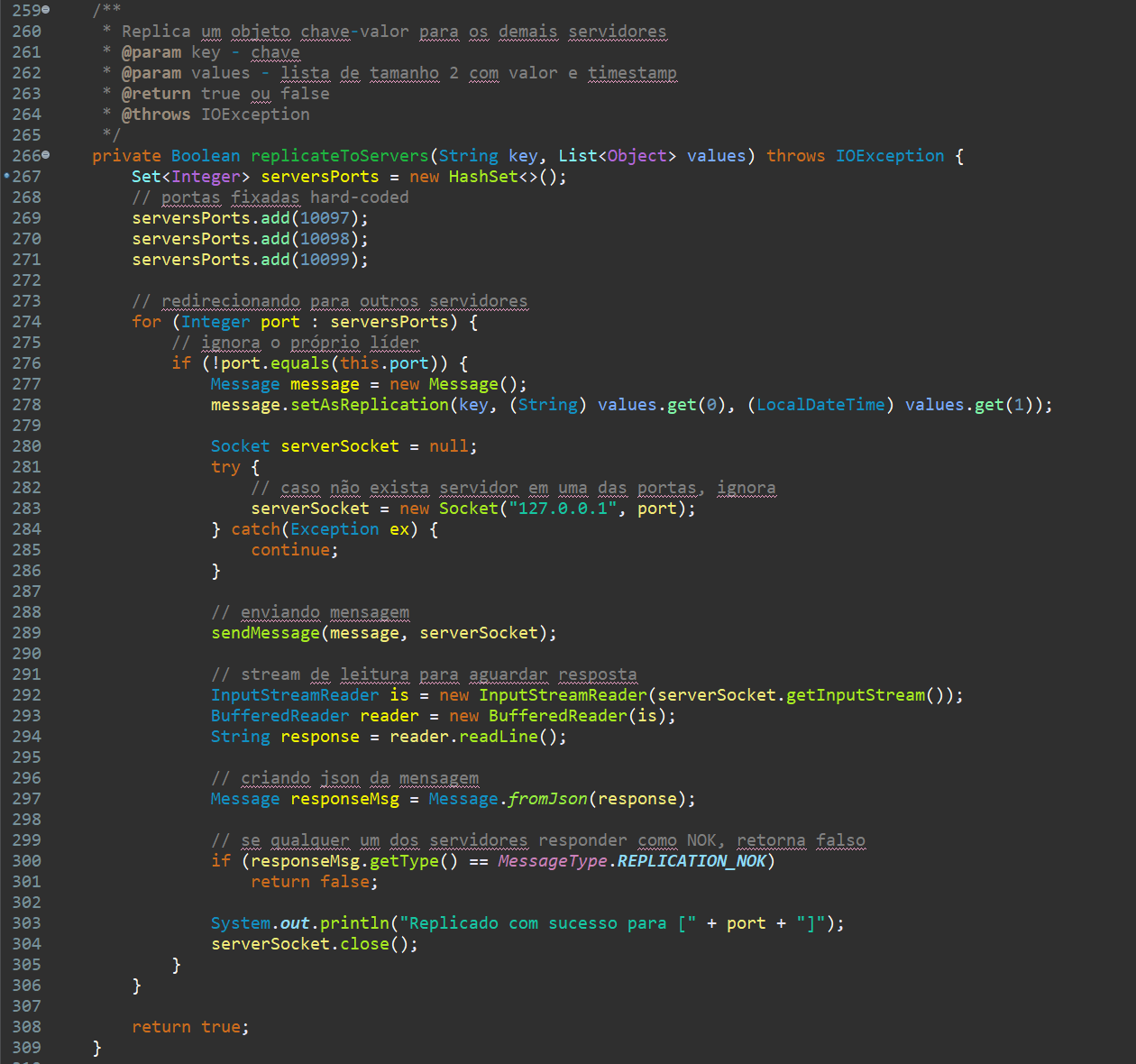


Figura método replicateToServers

**iAmLeader:** responsável por identificar se o servidor em questão é o líder ou não.

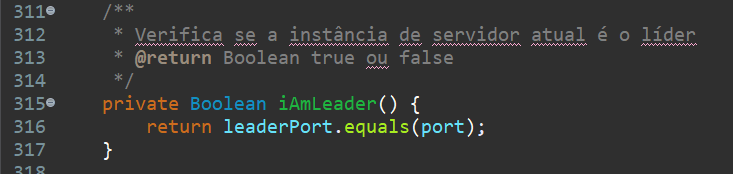


Figura método iAmLeader

**sendMessage:** responsável por enviar uma mensagem qualquer à um socket qualquer.

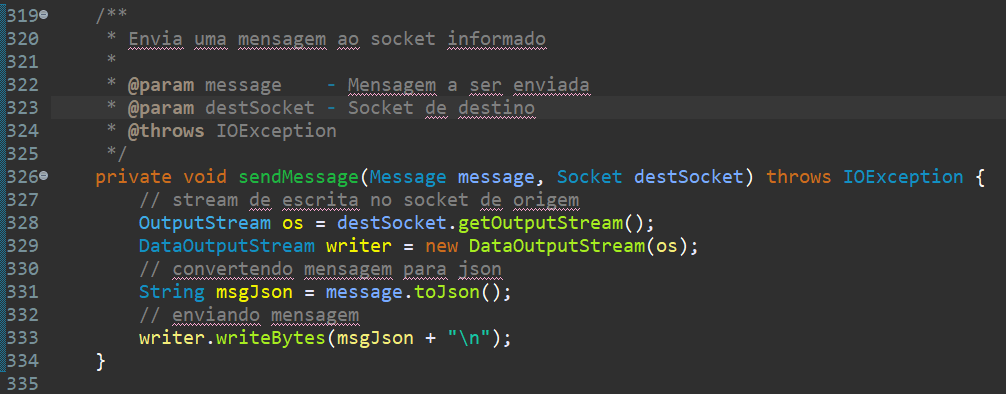


Figura método sendMessage