Nome: Daniel Espindola da Silva
RA: 11201720556

**Para compilar e rodar o server deve ser usado o seguinte comando:**

javac zookeeper/Message.java & javac -cp .;lib/gson-2.9.1.jar; zookeeper/Server.java & java -cp .;lib/gson-2.9.1.jar; zookeeper/Server

**Para compilar e rodar o client deve ser usado o seguinte comando:**

javac -cp .;lib/gson-2.9.1.jar; zookeeper/Client.java & java -cp .;lib/gson-2.9.1.jar; zookeeper/Client

Para ambos os casos é importante estar na raiz onde contém a pasta zookeeper com os java e a pasta lib com o gson.

Link para o vídeo com a compilação e execução do código:

https://www.youtube.com/watch?v=5xz9Nn28Ly4

## Funcionalidades do Client

**Inicialização** (linha 150)**:**
Na inicialização o client lê do terminal qual funcionalidade o usuário gostaria de executar, INIT, PUT ou GET, esse processo ocorre dentro de um while true, de modo que o usuário pode executar quantas vezes quiser as funcionalidades. Ao receber um INIT, é chamada a função obtemDadosTeclado (definida na linha 91), para a qual é passado o Scanner, essa função é responsável por ler 3 ips e portas do teclado, especificando o endereço dos servidores do sistema e salvando esses dados na memória.

**Envio de PUTs** (linha 158)**:**
Ao receber um PUT, o client captura do teclado a key e o value a serem inseridos, ambos são strings e ficam separados por espaço. Com essas informações é instanciado um objeto Message com tipo PUT e key e value e então esse objeto é passado para a ThreadPut, que enviará a mensagem para um servidor aleatório. A thread utiliza a função enviaMensagem, definida na linha 114, para sortear um servidor, iniciar o socket, converter a mensagem para string, enviar e retornar a resposta do servidor.

**Envio de GETs** (linha 175)**:**
O GET ocorre de maneira similar, é lido do teclado a key a ser procurada, e é instanciada uma classe mensagem com a key e o timestamp que o usuário tem da key, caso ele nunca tinha obtido essa Key, o timestamp vai como null. É então instanciada uma ThreadGet para enviar o get assincronamente e essa thread utiliza a função enviaMensagem da mesma maneira que o PUT.

## Funcionalidades do Servidor

**Inicialização** (linha 232)**:**

Ao inicializar, o servidor recebe da entrada padrão seu ip e porta , no formato ip:porta, e o ip e porta do servidor líder. Com essas informações ele determina os outros dois servidores conferindo as portas que restaram, dentre a 10097, 10098 e 10099. Nesse momento também ele já é capaz de dizer se é o líder ou não, pois ele sabe o ip e porta do líder para comparar com o seu ip e porta.

**Recebimento de PUTs** (linha 91)**:**
O recebimento de PUTs ocorre dentro da ThreadAtendimento, essa funcionalidade começa verificando se o servidor atual é o líder, isso é feito pela função souLider definida na linha 29. Caso o servidor seja o líder ele gera um timestamp atual, pegando o tempo do sistema em milissegundos e verifica se na hashtable do servidor já existe a key a ser inserida, caso exista ele substitui ela e atualiza o timestamp, caso não exista ele insere ela e o timestamp gerado. Depois disso ela gera uma mensagem de REPLICATION, colocando nessa a key, value e timestamp que foram inseridos e envia essa mensagem para os dois outros servidores do sistema, só depois de receber uma mensagem REPLICATION_OK de ambos os servidores ele cria uma nova Mensagem de PUT_OK e retorna para o client que havia solicitado a inserção.
Caso o servidor não seja o líder, ele encaminha a mensagem recebida para o líder e fica no aguardo do retorno dele para enviar o PUT_OK para o client.

**Recebimento de Replications** (linha 155)**:**
Ao receber um REPLICATION, o servidor insere ou substitui a key, value e timestamps nas suas hashtables e retorna um REPLICATION_OK para o servidor.

**Recebimentos de GETs** (linha 176)**:**
O recebimento de GET começa com o servidor verificando se a key procurada existe na sua hashtable, caso não exista o servidor retorna um value null na mensagem.
Caso a key solicitada exista, o servidor então verifica se o timestamp que ele tem para aquela key é mais recente que o do client que solicitou, caso seja, ele retorna o valor e o seu timestamp, caso não seja ele retorna o value “TRY_OTHER_SERVER_OR_LATER” para informar o client que o seu timestamp é mais recente.

## Uso de Threads

**Client:**
No client são utilizadas threads para o envio das requisições PUTs e GETs, de modo que o client pode executar outras funcionalidades enquanto aguarda o retorno da requisição.

A ThreadPut (linha 27) é responsável por executar o envio da mensagem PUT e colocar nos hashmaps os devidos valores e timestamps
A ThreadGet (linha 59) é responsável pelo envio do GET ao servidor, colocando nas variáveis o valor e timestamp caso a responsa do servidor não seja null.

**Servidor:**
A Thread mais importante do sistema é a ThreadAtendimento do servidor, essa Thread é instanciada na linha 253, toda vez que o servidor aceita a abertura de um socket, desse modo, o servidor consegue receber várias requisições em paralelo sem ficar travado em

uma requisição específica. A ThreadAtendimento recebe o socket obtido no serverSocket.accept() ao ser instanciada.
A definição dessa thread começa na linha 61. A função run da thread começa utilizando o socket para instanciar as classes de leitura e escrita para o socket, utilizando o BufferedReader lemos a string JSON recebida e convertemos ela na classe Message utilizando a biblioteca GSON (linha 80).
Na sequência verificamos o campo Tipo do objeto mensagem obtido para determinar qual ação deverá ser tomada, isso é feito num Switch Case na linha 84. O código então se divide em casos a depender do tipo de mensagem: PUT, REPLICATION e GET,  que foram explicados nas funcionalidades do servidor.
Ao fim dos tratamentos específicos de cada requisição, fechamos o socket e o método run da thread é finalizado.