Nome: Bruno Felix Dias - Matrícula: 428903
Nome: Paulo Ricardo da Silva Lopes - Matrícula: 385173

Questão 3

- Neste exercício temos a missão de criar um processo *Servidor* que salve o estado, ou seja, guarde as informações enviadas pelo processo *Cliente* e, em caso de queda de conexão, continue a execução do serviço do último ponto de interrupção.
  Para podermos armazenar os dados enviados pelo processo *Cliente*, criamos uma classe *Buffer* que identifica o *Cliente* através do seu *IP*. Nesta classe *Buffer* armazenamos também a operação e os operandos.
  Código:

```
public class Buffer {
        String ip;;
        double value1 = 0;
        double value2 = 0;
        String operator = "";

        public void setIp(String aIp) {
                this.ip = aIp;
        }
```

- No processo *Cliente* nós instanciamos um objeto do tipo *DataInputStream* para podermos ler a resposta do processo *Servidor*.

```
public class Client {
        public static void main(String[] args) {
                Socket client = null;
                DataInputStream in;
                String data;
                try {
                        client = new Socket("localhost", 33000);
                        in = new DataInputStream(client.getInputStream());
                        @SuppressWarnings("unused")
                        Send s = new Send(client);
                        while (true) {
                                data = in.readUTF();
                                System.out.println("Result: " + data);
                        }
                } catch (Exception e)
```

- Também criamos uma classe, a *Send*, que estendemos à classe *Thread*. Essa classe é basicamente para a troca de mensagens entre os processos.
  Código:

```
public class Send extends Thread {
        private static Scanner input = new Scanner(System.in);
        Socket aclient;
        DataOutputStream out;
        String data;

        public Send(Socket client) throws IOException {
                aclient = client;
                out = new DataOutputStream(aclient.getOutputStream());
                this.start();
```

```
11          }
12
13          public void run() {
14                  try {
15                          while (true) {
16                                  data = input.nextLine();
17                                  out.writeUTF(data);
18                          }
19                  } catch (Exception e)
```

- A função do processo *Servidor* é estabelecer a conexão com o processo *Cliente*, e dado que a conexão foi estabelecida ele verifica se esse *Cliente* já se conectou alguma vez alguma vez, caso não conste que ele tenha dados salvo no *Servidor*, ele cria um *Buffer* para esse novo *Cliente*.
  Código:

```
1  else {
2               System.out.println("Conex o estabelecida");
3                   Buffer b = new Buffer();
4                   b.setIp(String.valueOf(client.getInetAddress()));
5                   @SuppressWarnings("unused")
6                   ClientStates cs = new ClientStates(client, b);
7                   amz.add(b);
8           }
```

  Caso contrário, o *Servidor* recupera esses dados.
  Código:

```
1  if (amz.checking(String.valueOf(client.getInetAddress()))) {
2          System.out.println("Conex o restabelecida");
3          @SuppressWarnings("unused")
4          ClientStates cs = new ClientStates(client, amz.getBuffer(String.
               valueOf(client.getInetAddress())));
5  }
```

- Criamos uma classe para ser o nosso *array de Buffer*, a classe *Armazem*. Nesta classe temos dois *métodos*. Um deles é o *getBuffer*, que retorna os dados do *Cliente*.
  Código:

```
1  public Buffer getBuffer(String ip) {
2                   for (Buffer clientBuffer : amaz) {
3                           if (clientBuffer.ip.equals(ip)) {
4                                   return clientBuffer;
5                           }
```

  E o outro é o *método checking*, que verifica se já existem dados salvos do *Cliente*.
  Código:

```
1  public boolean checking(String ip) {
2                   for (Buffer clientBuffer : amaz) {
3                           if (clientBuffer.ip.equals(ip)) {
4                                   return true;
5                           }
6                   }
7                   return false;
8           }
```

- Uma outra classe também foi criada, a classe *ClientStates*, que é através dela que passamos os dados do *Cliente* para o *Buffer*. Nos parâmetros dessa classe nós passamos o *Cliente* e seu *Buffer*. (Construtor) Código:

```
public ClientStates(Socket s, Buffer b) throws IOException {
                client = s;
                buffer = b;
                in = new DataInputStream(client.getInputStream());
                out = new DataOutputStream(client.getOutputStream());
                this.start();
        }
```

- E por fim temos a classe *Received* que faz o recebimento dos dados.
Código:

```
public class Received {
        Socket client;
        DataInputStream in;
        String data;
        public Received(Socket s) throws IOException {
                this.client = s;
                in = new DataInputStream(client.getInputStream());
        }
        public void run() {
                try {
                        while(true) {
                                data = in.readUTF();
                                System.out.println(data);
                        }
                }catch (Exception e)...
```

Referências

[GUJ] https://www.guj.com.br/ - acessado em 20 set. 2019.

[Coulouris et al. 2013] Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2013). *Sistemas Distribuídos - Conceitos e Projetos*. Bookman Companhia Editora LTDA, 5ª edition.

[Tanenbaum and Steen 2008] Tanenbaum, A. S. and Steen, M. V. (2008). *Sistemas Distribuídos - Princípios e Paradigmas*. Pearson Education do Brasil, 2ª edition.