

PRÁTICA 1

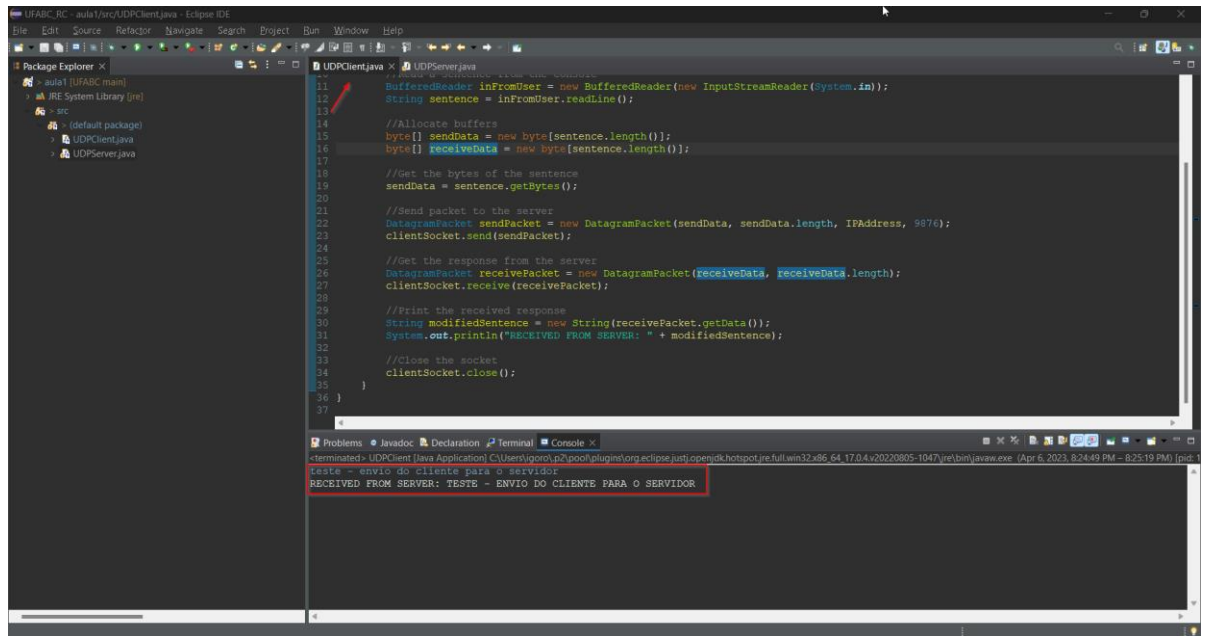
Nome: Igor Carvalho de Oliveira

RA: 11201920763

1. Aplicação cliente-servidor UDP

a. Execute primeiro o servidor (UDPServer.java) e depois o cliente (UDPClient.java). O que aconteceu? Justifique.

O servidor está pronto para receber dados do cliente ao executar o UDPServer. Assim que dados os dados forem enviados pelo cliente, o servidor estará preparado para tratar as informações recebidas e imprimir as informações recebidas sem tratamento. Além disso, o servidor imprime o IP do cliente (nesse caso, local host) que enviou a informação. Quando executamos o UDPClient, possibilitamos que usuários insiram entradas para serem disparadas ao Servidor. Neste caso, está sendo utilizado o LocalHost. Após enviar informações para o Servidor, o cliente recebe uma mensagem retornando à informação gravada (letras maiúsculas).

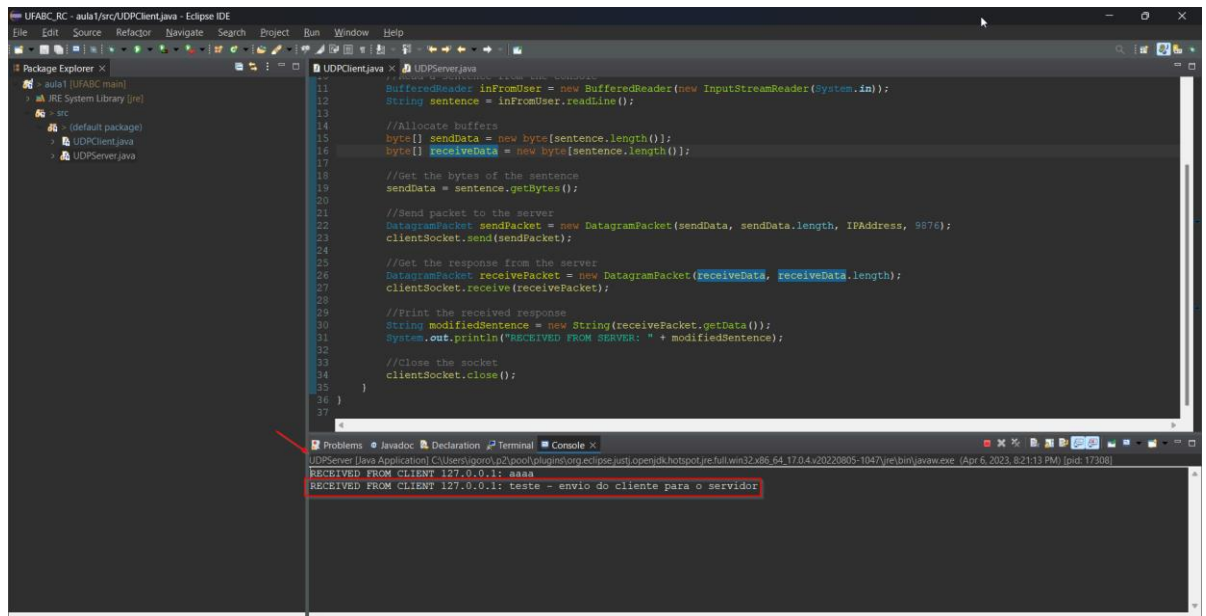


The screenshot shows the Eclipse IDE with the UDPServer.java file open. The code is as follows:

```
11 BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
12 String sentence = inFromUser.readLine();
13
14 //Allocate buffers
15 byte[] sendData = new byte[sentence.length()];
16 byte[] receiveData = new byte[sentence.length()];
17
18 //Get the bytes of the sentence
19 sendData = sentence.getBytes();
20
21 //Send packet to the server
22 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
23 clientSocket.send(sendPacket);
24
25 //Get the response from the server
26 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
27 clientSocket.receive(receivePacket);
28
29 //Print the received response
30 String modifiedSentence = new String(receivePacket.getData());
31 System.out.println("RECEIVED FROM SERVER: " + modifiedSentence);
32
33 //Close the socket
34 clientSocket.close();
35
36 }
37
```

The console output shows the following message:

```
<terminated> UDPClient [Java Application] C:\Users\igoro\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\java.exe (Apr 6, 2023, 8:24:49 PM) [pid: 1012]
teste - envio do cliente para o servidor
RECEIVED FROM SERVER: TESTE - ENVIO DO CLIENTE PARA O SERVIDOR
```



The screenshot shows the Eclipse IDE with the UDPServer.java file open. The code is the same as in the previous screenshot.

The console output shows the following message:

```
UDPServer [Java Application] C:\Users\igoro\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\java.exe (Apr 6, 2023, 8:21:13 PM) [pid: 17308]
RECEIVED FROM CLIENT 127.0.0.1: aaaa
RECEIVED FROM CLIENT 127.0.0.1: teste - envio do cliente para o servidor
```

b. Execute primeiro o cliente e depois o servidor. O que aconteceu? Justifique.

O cliente não recebe retorno do servidor, pois neste momento não há processos sendo executados do lado do servidor para gravar a informação recebida naquela porta. Neste caso, o pacote é descartado e o cliente fica bloqueado aguardando uma resposta.

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is as follows:

```
11 // created on 29/08/2019 at 22:53
12 // Import java.net.*;
13
14 //Allocate buffers
15 byte[] sendData = new byte[sentence.length()];
16 byte[] receiveData = new byte[sentence.length()];
17
18 //Get the bytes of the sentence
19 sendData = sentence.getBytes();
20
21 //Send packet to the server
22 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
23 clientSocket.send(sendPacket);
24
25 //Get the response from the server
26 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
27 clientSocket.receive(receivePacket);
28
29 //Print the received response
30 String modifiedSentence = new String(receivePacket.getData());
31 System.out.println("RECEIVED FROM SERVER: " + modifiedSentence);
32
33 //Close the socket
34 clientSocket.close();
35
36 }
```

The console window at the bottom shows three terminated processes:

```
1 <terminated> UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:07:06 PM - 8:28:33 PM) [pid: 19408]
2 <terminated> UDPServer [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:21:13 PM - 8:28:25 PM) [pid: 17308]
3 <terminated> UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:24:49 PM - 8:25:19 PM) [pid: 15668]
```

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is the same as in the previous screenshot. The console window at the bottom shows a message:

```
UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:30:08 PM) [pid: 21064]
teste - execução do client com servidor parado
```

The screenshot shows the Eclipse IDE with the UDPServer.java file open. The code is as follows:

```
1 // created on 29/08/2019 at 22:53
2 import java.net.*;
3
4 class UDPServer {
5     public static void main(String args[]) throws Exception {
6         //Create server socket
7         DatagramSocket serverSocket = new DatagramSocket(9876);
8         while(true) {
9             byte[] receiveData = new byte[1024];
10            //block until packet is sent by client
11            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
12            serverSocket.receive(receivePacket);
13            //Get the information about the destination of the client
14            InetAddress IPAddress = receivePacket.getAddress();
15            int port = receivePacket.getPort();
16            //Get the data of the packet
17            String sentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
18            System.out.println("RECEIVED FROM CLIENT: "+IPAddress.getHostAddress()+" : "+ sentence);
19            //Change the data to capital letters
20            String capitalizedSentence = sentence.toUpperCase();
21            byte[] sendData = new byte[sentence.length()];
22            sendData = capitalizedSentence.getBytes();
23            //Send back the response to the client
24            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
25            serverSocket.send(sendPacket);
26        }
27    }
28 }
```

The console window at the bottom shows two running processes:

```
1 UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:30:08 PM) [pid: 21064]
2 UDPServer [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:31:17 PM) [pid: 20148]
```

c. Altere as portas do servidor e cliente. O que acontece se as portas forem diferentes? Justifique.

Ao alterar as portas do Servidor e Cliente de modo que elas sejam diferentes entre si, notamos que o cliente não recebe informação de retorno do servidor, pois nesse caso a informação foi enviada para um endereço diferente da porta do servidor, ou seja, a informação foi enviada para outro endereço. Se a porta for reajustada no cliente, o envio será feito corretamente e o servidor irá retornar uma mensagem conforme evidenciado na última imagem.

```

1 // created on 29/09/2010 at 22:33
2 import java.net.*;
3
4 class UDPServer {
5     public static void main(String args[]) throws Exception {
6         //Create server socket
7         DatagramSocket serverSocket = new DatagramSocket(9000);
8
9         while(true) {
10             byte[] receiveData = new byte[1024];
11             //block until packet is sent by client
12             DatagramPacket receivedPacket = new DatagramPacket(receiveData, receiveData.length);
13             serverSocket.receive(receivedPacket);
14
15             //Get the information about the datagram of the client
16             InetAddress IPAddress = receivedPacket.getAddress();
17             int port = receivedPacket.getPort();
18
19             //Get the data of the packet
20             String sentence = new String(receivedPacket.getData(), 0, receivedPacket.getLength());
21             System.out.println("RECEIVED FROM CLIENT "+IPAddress.getHostAddress()+" : " + sentence);
22
23             //Change the data to capital letters
24             String capitalizedSentence = sentence.toUpperCase();
25             byte[] sendData = new byte[sentence.length()];
26             sendData = capitalizedSentence.getBytes();
27

```

The screenshot shows the Eclipse IDE with two Java files open: UDPServer.java and UDPClient.java. The UDPServer.java file is the same as shown in the previous image. The UDPClient.java file is shown with the following code:

```

1 import java.io.*;
2
3 class UDPClient {
4     public static void main(String args[]) throws Exception {
5         //Create datagram socket
6         DatagramSocket clientSocket = new DatagramSocket();
7         InetAddress IPAddress = InetAddress.getByName("localhost");
8
9         //Read a sentence from the console
10        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
11        String sentence = inFromUser.readLine();
12
13        //Allocate buffers
14        byte[] sendData = new byte[sentence.length()];
15        byte[] receiveData = new byte[sentence.length()];
16
17        //Get the bytes of the sentence
18        sendData = sentence.getBytes();
19
20        //Send packet to the server
21        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
22        clientSocket.send(sendPacket);
23
24        //Get the response from the server
25        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
26        clientSocket.receive(receivePacket);
27
28    }
29 }

```

The console output shows the server receiving a message from the client:

```

[JPClient [Java Application] C:\Users\jgorro.p2\workspace\org.eclipse.jdt.openjdk.hotspot.jre.full\win32_x86_64_17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:36:25 PM) [pid: 10372]
teste - envio de pacote tendo mudança de porta do lado do servidor

```

The screenshot shows the Eclipse IDE with the UDPServer.java file open. The code is the same as shown in the previous image. The console output shows the server receiving a message from the client:

```

[JPClient [Java Application] C:\Users\jgorro.p2\workspace\org.eclipse.jdt.openjdk.hotspot.jre.full\win32_x86_64_17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:35:43 PM) [pid: 10188]
teste - envio de pacote tendo mudança de porta do lado do servidor

```

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is as follows:

```
10 //Read a sentence from the console
11 BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
12 String sentence = inFromUser.readLine();
13
14 //Allocate buffers
15 byte[] sendData = new byte[sentence.length()];
16 byte[] receiveData = new byte[sentence.length()];
17
18 //Get the bytes of the sentence
19 sendData = sentence.getBytes();
20
21 //Send packet to the server
22 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9000);
23 clientSocket.send(sendPacket);
24
25 //Get the response from the server
26 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
27 clientSocket.receive(receivePacket);
28
29 //Print the received response
30 String modifiedSentence = new String(receivePacket.getData());
31 System.out.println("RECEIVED FROM SERVER: " + modifiedSentence);
32
33 //Close the socket
34 clientSocket.close();
35 }
36 }
```

The console output shows:

```
<terminated> UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:40:08 PM) [pid: 20952]
teste - envio com porta ajustada
RECEIVED FROM SERVER: TESTE - ENVIO COM PORTA AJUSTADA
```

d. Execute o servidor e cliente em máquinas diferentes. Em seguida, envie várias mensagens ao servidor a partir de mais de um cliente (várias máquinas diferentes) em execução. O que aconteceu?

O servidor recebeu e gravou informações recebidas de clientes diferentes, não sendo restrito o seu uso por apenas um cliente específico. Como eu não tenho máquinas diferentes, acabei simulando clientes diferentes dentro do meu próprio código, alterando apenas a descrição de cada envio.

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is the same as in the previous screenshot. The console output shows:

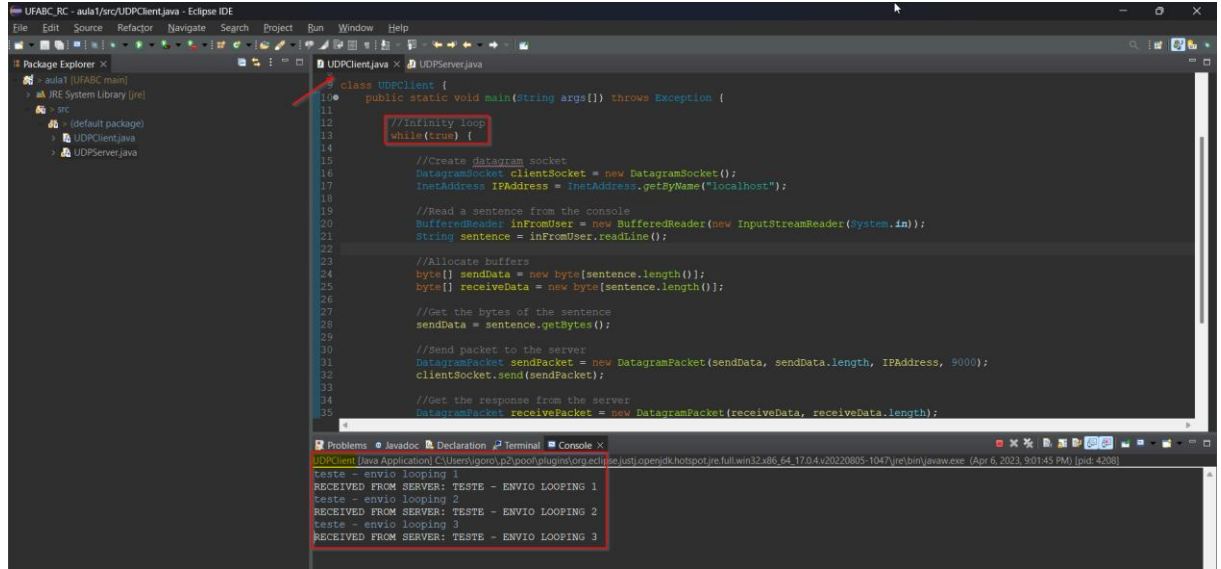
```
UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:45:58 PM) [pid: 20952]
teste - envio do cliente1
```

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is the same as in the previous screenshots. The console output shows:

```
UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\javaw.exe (Apr 6, 2023, 8:45:15 PM) [pid: 16056]
RECEIVED FROM CLIENT 127.0.0.1: teste - envio do cliente1
RECEIVED FROM CLIENT 127.0.0.1: teste - envio do cliente2
RECEIVED FROM CLIENT 127.0.0.1: teste - envio do cliente3
RECEIVED FROM CLIENT 127.0.0.1: teste - envio do cliente4
RECEIVED FROM CLIENT 127.0.0.1: teste - envio do cliente5
```


- e. Modifique o programa cliente para enviar mais de uma mensagem digitada pelo usuário. E necessário alterar também o código do servidor? Justifique.

Não. É apenas necessário incluir o comando **while(true)** para que o programar rode mais do que uma vez no cliente a fim de enviar/receber informações do servidor novamente a partir de uma única execução. Nesse caso, o comando **while(true)** cria um looping infinito no código permitindo que o cliente possa inserir N vezes uma informação a ser enviada ao servidor. Vale ressaltar que esse comando gera um looping infinito, então será necessário interromper o programa para impedir que o usuário realize uma entrada de dados. O servidor em si já está preparado para receber N informações de N clientes de maneira simultânea e por isso não foi necessário alterar o seu código.



```
class UDPClient {
    public static void main(String args[]) throws Exception {
        //Infinity loop
        while(true) {
            //Create datagram socket
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress IPAddress = InetAddress.getByName("localhost");

            //Read a sentence from the console
            BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
            String sentence = inFromUser.readLine();

            //Allocate buffers
            byte[] sendData = new byte[sentence.length()];
            byte[] receiveData = new byte[sentence.length()];

            //Get the bytes of the sentence
            sendData = sentence.getBytes();

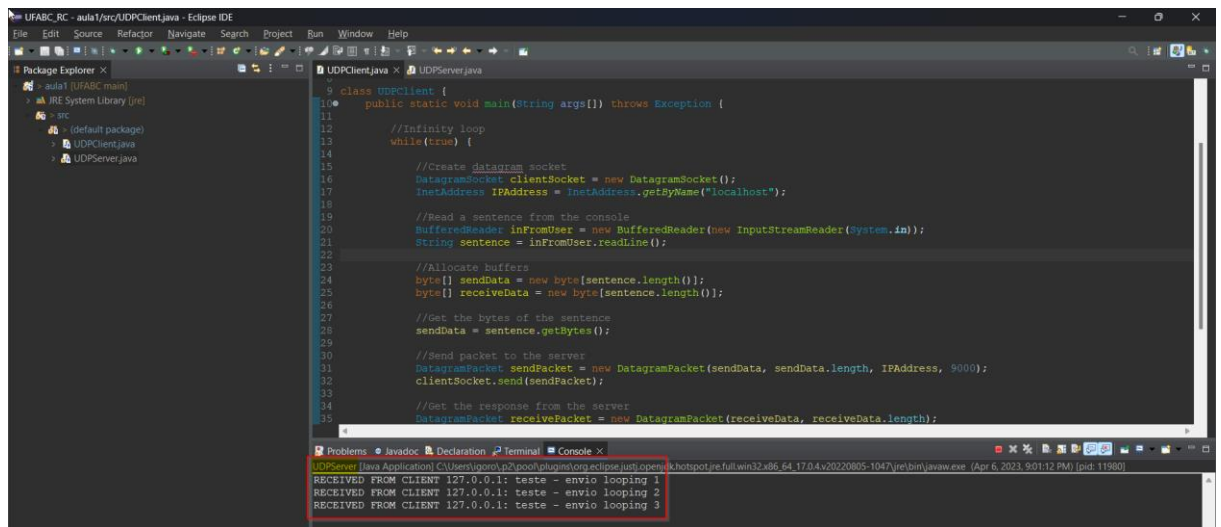
            //Send packet to the server
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9000);
            clientSocket.send(sendPacket);

            //Get the response from the server
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            clientSocket.receive(receivePacket);

            //Print the response
            System.out.println("RECEIVED FROM SERVER: " + new String(receiveData));
        }
    }
}
```

Console Output:

```
teste - envio looping 1
RECEIVED FROM SERVER: TESTE - ENVIO LOOPING 1
teste - envio looping 2
RECEIVED FROM SERVER: TESTE - ENVIO LOOPING 2
teste - envio looping 3
RECEIVED FROM SERVER: TESTE - ENVIO LOOPING 3
```



```
class UDPClient {
    public static void main(String args[]) throws Exception {
        //Infinity loop
        while(true) {
            //Create datagram socket
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress IPAddress = InetAddress.getByName("localhost");

            //Read a sentence from the console
            BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
            String sentence = inFromUser.readLine();

            //Allocate buffers
            byte[] sendData = new byte[sentence.length()];
            byte[] receiveData = new byte[sentence.length()];

            //Get the bytes of the sentence
            sendData = sentence.getBytes();

            //Send packet to the server
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9000);
            clientSocket.send(sendPacket);

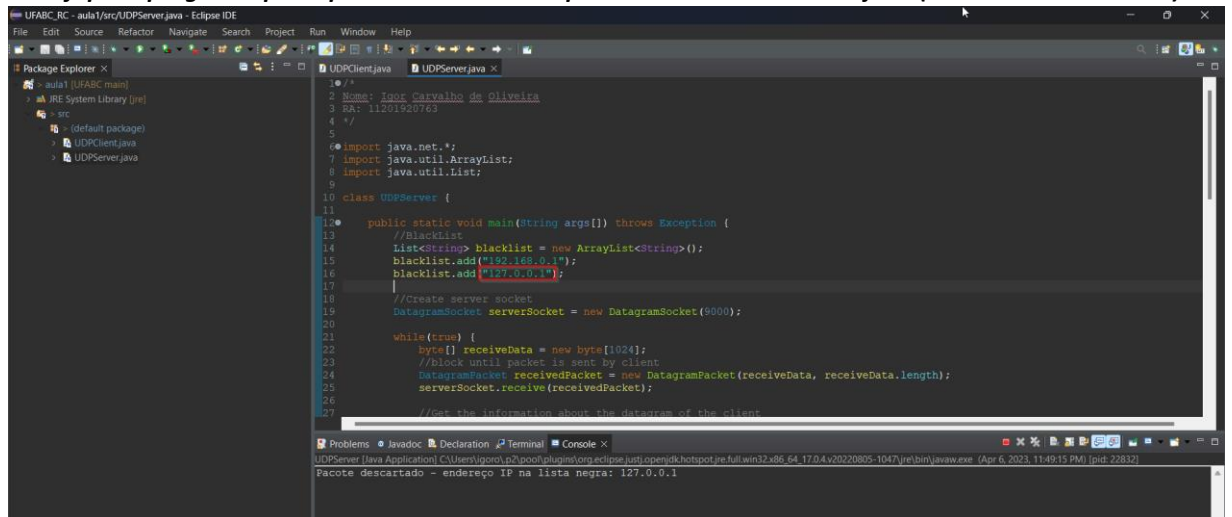
            //Get the response from the server
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            clientSocket.receive(receivePacket);

            //Print the response
            System.out.println("RECEIVED FROM CLIENT " + IPAddress + ": " + new String(receiveData));
        }
    }
}
```

Console Output:

```
RECEIVED FROM CLIENT 127.0.0.1: teste - envio looping 1
RECEIVED FROM CLIENT 127.0.0.1: teste - envio looping 2
RECEIVED FROM CLIENT 127.0.0.1: teste - envio looping 3
```

f. **Modifique o programa para que o servidor não responda a um usuário indesejado (White list ou Black list).**

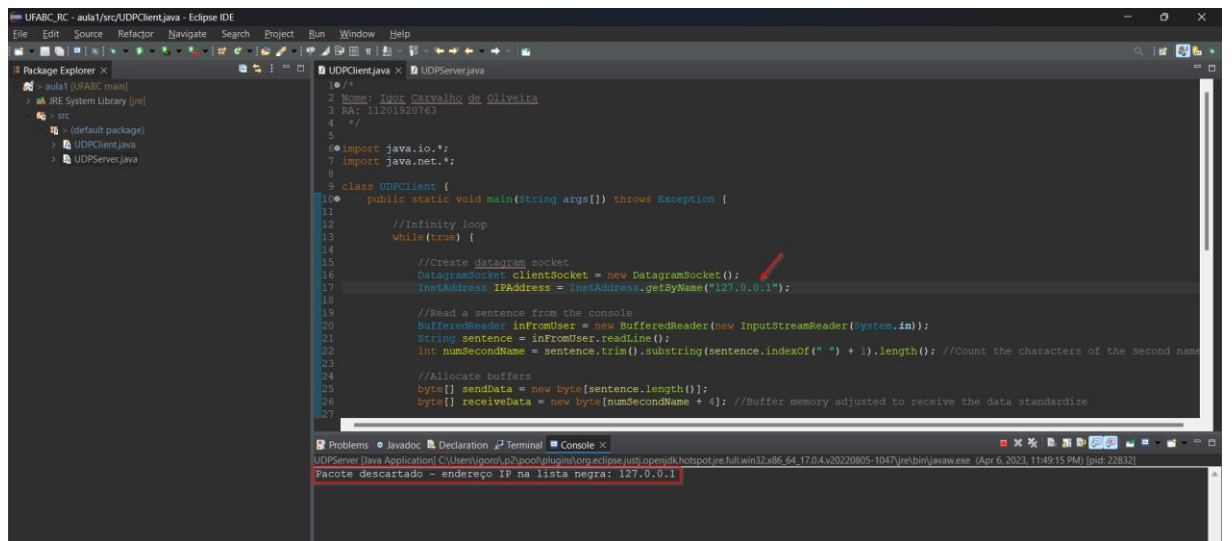


```
1 //
2 Nome: Igor Carvalho de Oliveira
3 RA: 11201920763
4 */
5
6 import java.net.*;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 class UDPServer {
11
12     public static void main(String args[]) throws Exception {
13         //Blacklist
14         List<String> blacklist = new ArrayList<String>();
15         blacklist.add("192.168.0.1");
16         blacklist.add("127.0.0.1");
17
18         //Create server socket
19         DatagramSocket serverSocket = new DatagramSocket(9000);
20
21         while(true) {
22             byte[] receiveData = new byte[1024];
23             //block until packet is sent by client
24             DatagramPacket receivedPacket = new DatagramPacket(receiveData, receiveData.length);
25             serverSocket.receive(receivedPacket);
26             //Get the information about the datagram of the client
27
28         }
29     }
30 }
```

Problems • Javadoc • Declaration • Terminal • Console X

UDPServer (Java Application) C:\Users\igor\p2\poo\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\javaw.exe (Apr 6, 2023, 11:49:15 PM) [pid: 22832]

Pacote descartado - endereço IP na lista negra: 127.0.0.1



```
1 //
2 Nome: Igor Carvalho de Oliveira
3 RA: 11201920763
4 */
5
6 import java.io.*;
7 import java.net.*;
8
9 class UDPClient {
10
11     public static void main(String args[]) throws Exception {
12         //Infinity loop
13         while(true) {
14
15             //Create datagram socket
16             DatagramSocket clientSocket = new DatagramSocket();
17             InetAddress IPAddress = InetAddress.getByName("127.0.0.1");
18
19             //Read a sentence from the console
20             BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
21             String sentence = inFromUser.readLine();
22             int numSecondName = sentence.trim().substring(sentence.indexOf(" ") + 1).length(); //Count the characters of the second name
23
24             //Allocate buffers
25             byte[] sendData = new byte[sentence.length()];
26             byte[] receiveData = new byte[numSecondName + 4]; //Buffer memory adjusted to receive the data standardize
27
28         }
29     }
30 }
```

Problems • Javadoc • Declaration • Terminal • Console X

UDPServer (Java Application) C:\Users\igor\p2\poo\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\javaw.exe (Apr 6, 2023, 11:49:15 PM) [pid: 22832]

Pacote descartado - endereço IP na lista negra: 127.0.0.1

2. Desenvolvimento de aplicação cliente-servidor UDP.

a. Crie um protocolo de aplicação qualquer e implemente o cliente e servidor para esse protocolo.

Imagine o contexto em que estamos implantando um novo sistema de monitoramento para uma empresa. Inicialmente, sabemos que os colaboradores já possuíam um login no antigo sistema (totalmente despadronizado) e o nosso desafio diante nesse cenário é garantir que o cadastro dos novos logins no novo sistema siga um padrão estabelecido pela empresa contratante.

Diante desse contexto, foi criado um protocolo visando criar e padronizar o username de acordo com o nome e sobrenome de uma pessoa. Dessa forma, o usuário irá informar seu nome e sobrenome e o servidor irá gerar um username de modo que todos os cadastros tenham sempre a mesma estrutura.

Todo username será composto por:

- Prefixo "RC_" + 1ª letra do Nome + Sobrenome
- O username sempre estará em letras maiúsculas.
- Exemplo: **IGOR OLIVEIRA** → **RC_OLIVEIRA**.

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is as follows:

```
25 //Allocate buffers
26 byte[] sendData = new byte[sentence.length()];
27 byte[] receiveData = new byte[numSecondName+1];
28
29 //Get the bytes of the sentence
30 sendData = sentence.getBytes();
31
32 //Send packet to the server
33 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9000);
34 clientSocket.send(sendPacket);
35
36 //Get the response from the server
37 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
38 clientSocket.receive(receivePacket);
39
40 //Print the received response
41 System.out.println("Received response: " + new String(receivePacket.getData()));
```

The console output shows the following sequence of events:

```
UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\java.exe (Apr 6, 2023, 11:22:51 PM) [pid: 8692]
igor oliveira
RECEIVED FROM SERVER
Submissions received by server has 13 characters.
The prefix 'RC_' will be included to standardize the data.
Update: RC_OLIVEIRA containing 12 characters.
carlos eduardo
RECEIVED FROM SERVER
Submissions received by server has 14 characters.
The prefix 'RC_' will be included to standardize the data.
Update: RC_CEDUARDO containing 11 characters.
gustavo pavani
RECEIVED FROM SERVER
Submissions received by server has 14 characters.
The prefix 'RC_' will be included to standardize the data.
Update: RC_GPAVANI containing 10 characters.
```

The screenshot shows the Eclipse IDE with the UDPClient.java file open. The code is as follows:

```
25 //Allocate buffers
26 byte[] sendData = new byte[sentence.length()];
27 byte[] receiveData = new byte[numSecondName+1];
28
29 //Get the bytes of the sentence
30 sendData = sentence.getBytes();
31
32 //Send packet to the server
33 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9000);
34 clientSocket.send(sendPacket);
35
36 //Get the response from the server
37 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
38 clientSocket.receive(receivePacket);
39
40 //Print the received response
41 System.out.println("Received response: " + new String(receivePacket.getData()));
```

The console output shows the following sequence of events:

```
UDPClient [Java Application] C:\Users\igor\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.4.v20220805-1047\jre\bin\java.exe (Apr 6, 2023, 11:22:42 PM) [pid: 22016]
RECEIVED FROM CLIENT 127.0.0.1:
Default login = RC_OLIVEIRA
RECEIVED FROM CLIENT 127.0.0.1:
Default login = RC_CEDUARDO
RECEIVED FROM CLIENT 127.0.0.1:
Default login = RC_GPAVANI
```