

Laboratório nº: 2

Data: sexta-feira, 10 de abril de 2020

Turma: 61D

Grupo: **Grupo 06**

Número e nome dos alunos presentes:

Número		Nome
43861		Francisco Chicharro
43874		João Florentino
Click or tap here to enter text.		Click or tap here to enter text.

1. Objetivo da atividade (descrição por palavras simples do que entendeu como objetivo da atividade);
 - ➔ Desenvolver serviço distribuído com tecnologia Java Remote Method Invocation (RMI)
 - ➔ Desenvolver serviço distribuído com tecnologia Google Remote Procedure Call (gRPC)
 - ➔ Comparar as tecnologias RMI e gRPC, em particular a capacidade de comunicação bidireccional

2. Indicação das tecnologias e as ferramentas (*tools*) utilizadas;

-Google Cloud Platform para criação da VM;

-Bitwise SSH Client que permite o estabelecimento de uma sessão SSH para comunicação;

-IntelliJ para executar as ações pretendidas através de código fornecido pelo Professor;

3. Descrição da arquitetura das partes (componentes) envolvidas, com eventuais diagramas:

Neste laboratório desenvolveu-se um serviço para cálculo de números primos, usando o serviço RMI. Um cliente do serviço pode enviar pedidos de cálculo indicando o número inicial e a quantidade de valores a retornar. Os serviços retornam, de forma assíncrona, a sequência de números primos.

Exercício 1

Alínea a)

Para a implementação da interface IPrimesService e do código RMI, manteve-se intacta a estrutura do código fornecido, alterando apenas alguns nomes dos objetos para adequar à solução. Implementou-se o método findPrimes.

Alínea b)

Através da interface ICallBack do código fornecido que tem o cliente RMI, procedeu-se à definição do método nextPrime que está na interface ICallBack que o cliente vai implementar (o método recebe um número primo).

Alínea c)

Como pedido no enunciado, pretende-se que um cliente chame várias vezes um serviço e que seja possível distinguir as respostas que o servidor para cada chamada.

É utilizado um único Stub para o cliente fazer as chamadas, permitindo uma melhor gestão dos recursos de memória.

Assim, optou-se por modificar os contratos IPrimeService e ICallback , estes vão receber um parâmetro extra chamado callID, de forma a ser possível distinguir as respostas de cada chamada.

Consequentemente, foi necessário alterar no servidor e cliente a definição dos métodos nextPrimes e findPrimes, no cliente para receber o novo parâmetro callID, no servidor para este conseguir imprimir o número primo.

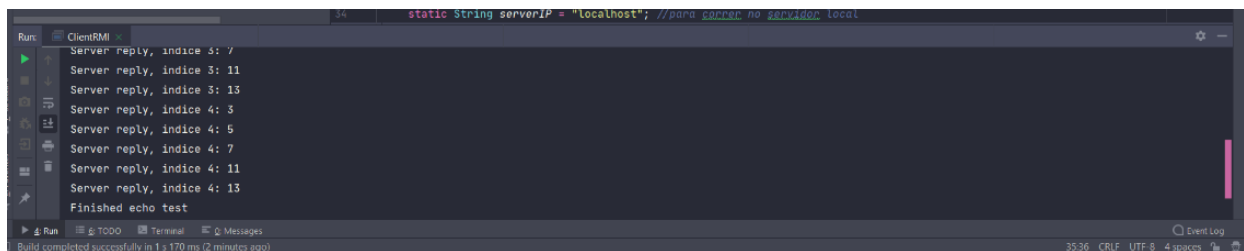
Para permitir ao cliente realizar múltiplas chamadas, é criado um objeto stub, onde através de um ciclo com limite, a cada iteração é chamado o método findPrimes onde para além dos parâmetros anteriores é adicionado o índice usado no ciclo para ordenar cada chamada.

O método findPrimes recebe como parâmetros um inteiro start, o número de número primos que se quer imprimir na consola a partir do inteiro start e o callback do cliente. Na implementação desta função utiliza-se o método auxiliar isPrime que testa se o número é primo. Se o for, chama-se o método nextPrime que mostra no terminal o número primo passado no parâmetro.

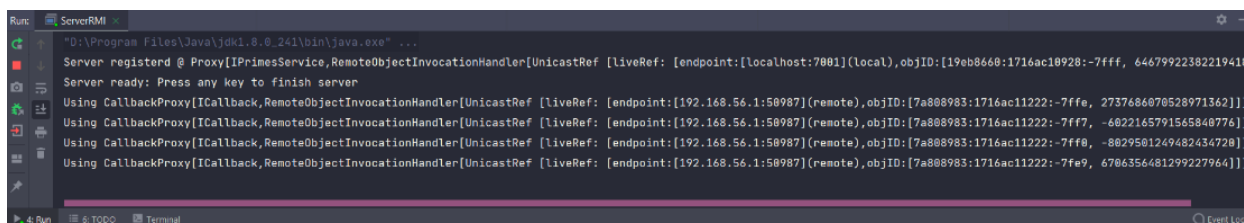
Nesta implementação, é de notar a invocação de uma Thread através de Thread.sleep(1000) sendo que esta suspende a execução da tarefa durante um período específico de tempo (neste caso 1 segundo).

Assim, após realizadas as alterações necessárias, correu-se o cliente onde o índice identifica várias chamadas diferentes, chamando o mesmo serviço várias vezes separadamente através do método findPrimes:

Cliente:



Servidor:



Alínea d)

Realizou-se três cenários entre um cliente um servidor RMI:

1) Cliente e servidor local;

2) Cliente e servidor numa VM da GCP;

3) Cliente e servidor em VMS diferentes da GCP usando os IPs internos das VMs.

No cenário 1), o cliente e o servidor correm sobre a mesma máquina sendo que o cliente liga-se ao servidor através do seu localhost. Para tal realizou-se uma chamada ao serviço onde o teste passou com sucesso:

Cliente:

```
"D:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Callback registered @ Proxy[ICallback,RemoteObjectInvocationHandler[UnicastRef [LiveRef: [endpoint:[192.168.56.1:51197]](local),objID:[3e2ccf9a:1716ac5a763:-7ffe, 8816862832413385]]]]
Server reply:3
Server reply:5
Server reply:7
Server reply:11
Server reply:13
Finished echo test

Process finished with exit code 0
```

Servidor:

```
ServerRMI
"D:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Server registered @ Proxy[IPrimesService,RemoteObjectInvocationHandler[UnicastRef [LiveRef: [endpoint:[localhost:7001]](local),objID:[-3e020464:1716ac4cd47:-7fff, -740067164630805]]]]
Server ready: Press any key to finish server
Using CallbackProxy[ICallback,RemoteObjectInvocationHandler[UnicastRef [LiveRef: [endpoint:[192.168.56.1:51197]](remote),objID:[3e2ccf9a:1716ac5a763:-7ffe, 8816862832413389793]]]]
```

No cenário 2) Enquanto o cliente está a correr localmente, o servidor corre numa máquina virtual. Uma vez que o cliente nem sempre tem controlo do endereço IP que é passado ao servidor através do callback. Mesmo que tivesse esse controlo, quando o IP do cliente passa para o exterior este irá ser manipulado através de protocolos externos (como o NAT) do fornecedor de serviços da Internet. Assim, é possível perceber o porquê deste teste não ter tido sucesso, uma vez que a ligação entre o cliente e o servidor não se estabelece:

```
ClientRMI
"D:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
java.rmi.ConnectException: Connection refused to host: 10.128.0.2; nested exception is:
  java.net.ConnectException: Connection timed out: connect <5 internal calls>
    at client.ClientRMI.main(ClientRMI.java:41)
Caused by: java.net.ConnectException: Connection timed out: connect
    at java.net.DualStackPlainSocketImpl.connect0(Native Method)
    at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:79)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:286)
```

No cenário 3) O cliente e o servidor correm em máquinas virtuais diferentes, estas que foram criadas na GCP, onde o servidor estava á escuta nos portos 80 e 81. O cliente fez um pedido ao IP interno do servidor (que ficou com o IP 10.128.0.2). Verificou-se o bom funcionamento uma vez que apesar do cliente e servidor estarem em máquinas distintas eles estavam a correr na mesma rede e assim, o servidor tinha a capacidade de responder ao IP interno do cliente sem este necessitar de ser traduzido por protocolos de rede externos (como o NAT).

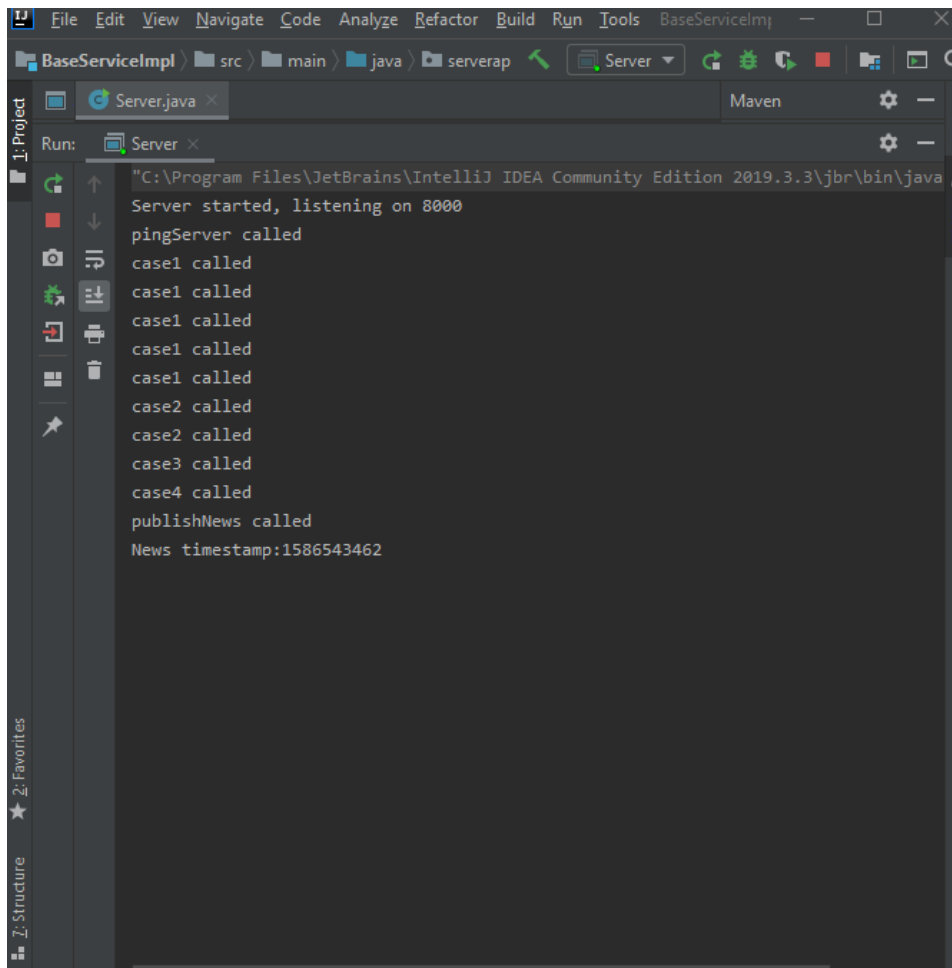
Servidor do lado esquerdo e cliente do lado direito:

```
G06-LI61D@146.148.91.140:22 - Bitvise xterm - G06-LI61D@maquinavirtualgrupo6: ~
G06-LI61D@maquinavirtualgrupo6:~$ sudo java -jar ServerRMI.jar 10.128.0.2
Server registered @ Proxy[IPrimesService,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[10.128.0.2:7001](local),objID:[1bd0f893:1716abd0aa7:-7fff, -696426615416346931]]]]]
Server ready: Press any key to finish server
Using CallbackProxy[ICallback,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[10.128.0.3:39915](remote),objID:[-4664214c:1716abdae4a:-7ffe, 7825845873840020556]]]]]
Using CallbackProxy[ICallback,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[10.128.0.3:39915](remote),objID:[-4664214c:1716abdae4a:-7fff, -7025756442921721711]]]]]
Using CallbackProxy[ICallback,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[10.128.0.3:39915](remote),objID:[-4664214c:1716abdae4a:-7ffe, 5440112805135466120]]]]]
Using CallbackProxy[ICallback,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[10.128.0.3:39915](remote),objID:[-4664214c:1716abdae4a:-7ffe, 5187250454547812851]]]]]

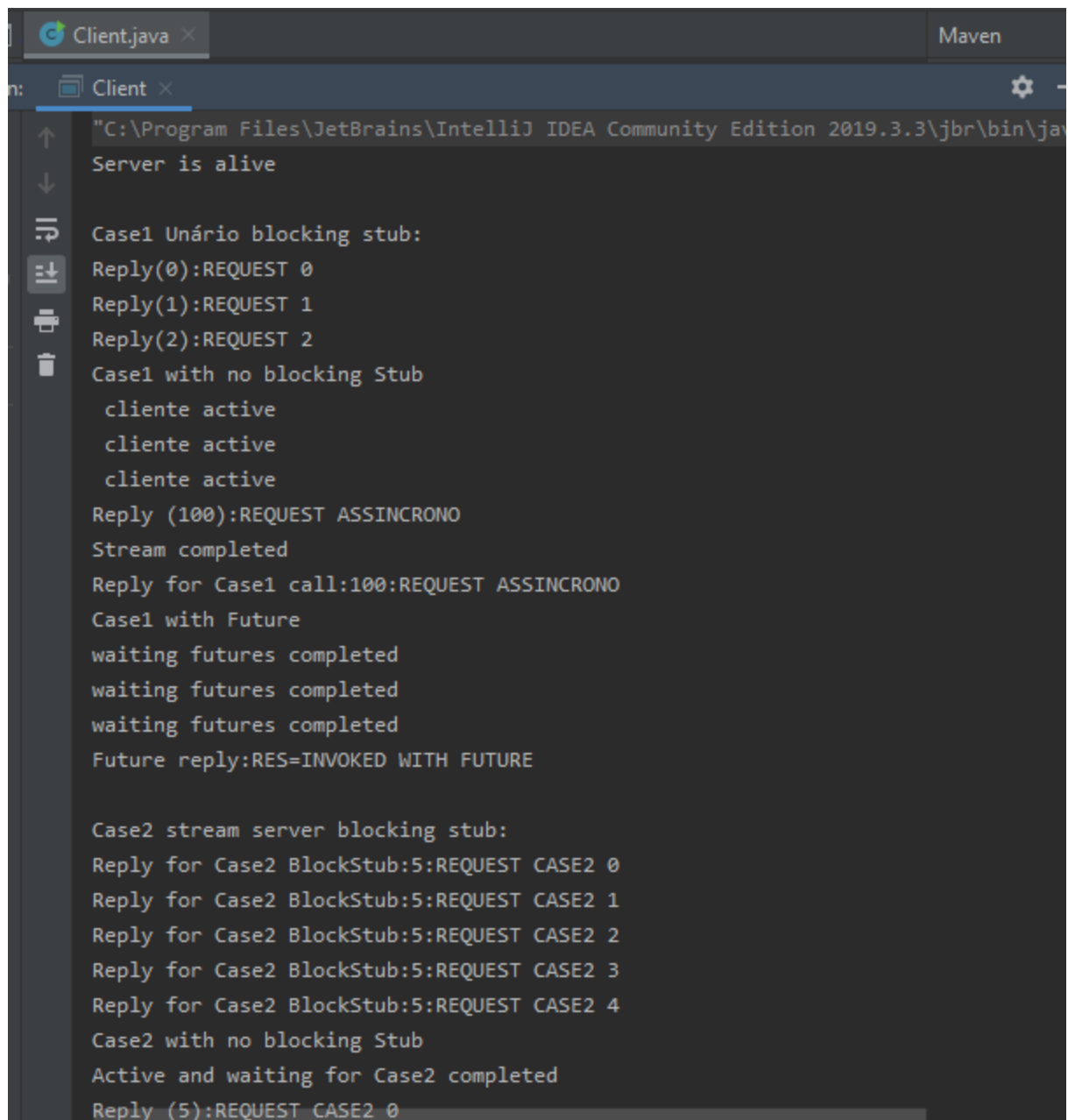
G06-LI61D@vmgrupo6:~$ sudo java -jar clientRMI.jar
Server reply, indice 1: 3
Server reply, indice 1: 5
Server reply, indice 1: 7
Server reply, indice 1: 11
Server reply, indice 1: 13
Server reply, indice 2: 3
Server reply, indice 2: 5
Server reply, indice 2: 7
Server reply, indice 2: 11
Server reply, indice 2: 13
Server reply, indice 3: 3
Server reply, indice 3: 5
Server reply, indice 3: 7
Server reply, indice 3: 11
Server reply, indice 3: 13
Server reply, indice 4: 3
Server reply, indice 4: 5
Server reply, indice 4: 7
Server reply, indice 4: 11
Server reply, indice 4: 13
Finished echo test
G06-LI61D@vmgrupo6:~$
```

Exercicio 2

Seguindo o enunciado do exercicio 2, copiou-se o artefacto Gerado no projeto rpcContract para a diretorio lib, de seguida iniciou-se o servidor e o cliente onde se testou a ligação, tendo esta sucesso:



```
File Edit View Navigate Code Analyze Refactor Build Run Tools BaseServiceImpl
BaseServiceImpl src main java serverap Server Maven
Run: Server
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\jbr\bin\java
Server started, listening on 8000
pingServer called
case1 called
case1 called
case1 called
case1 called
case1 called
case2 called
case2 called
case3 called
case4 called
publishNews called
News timestamp:1586543462
```

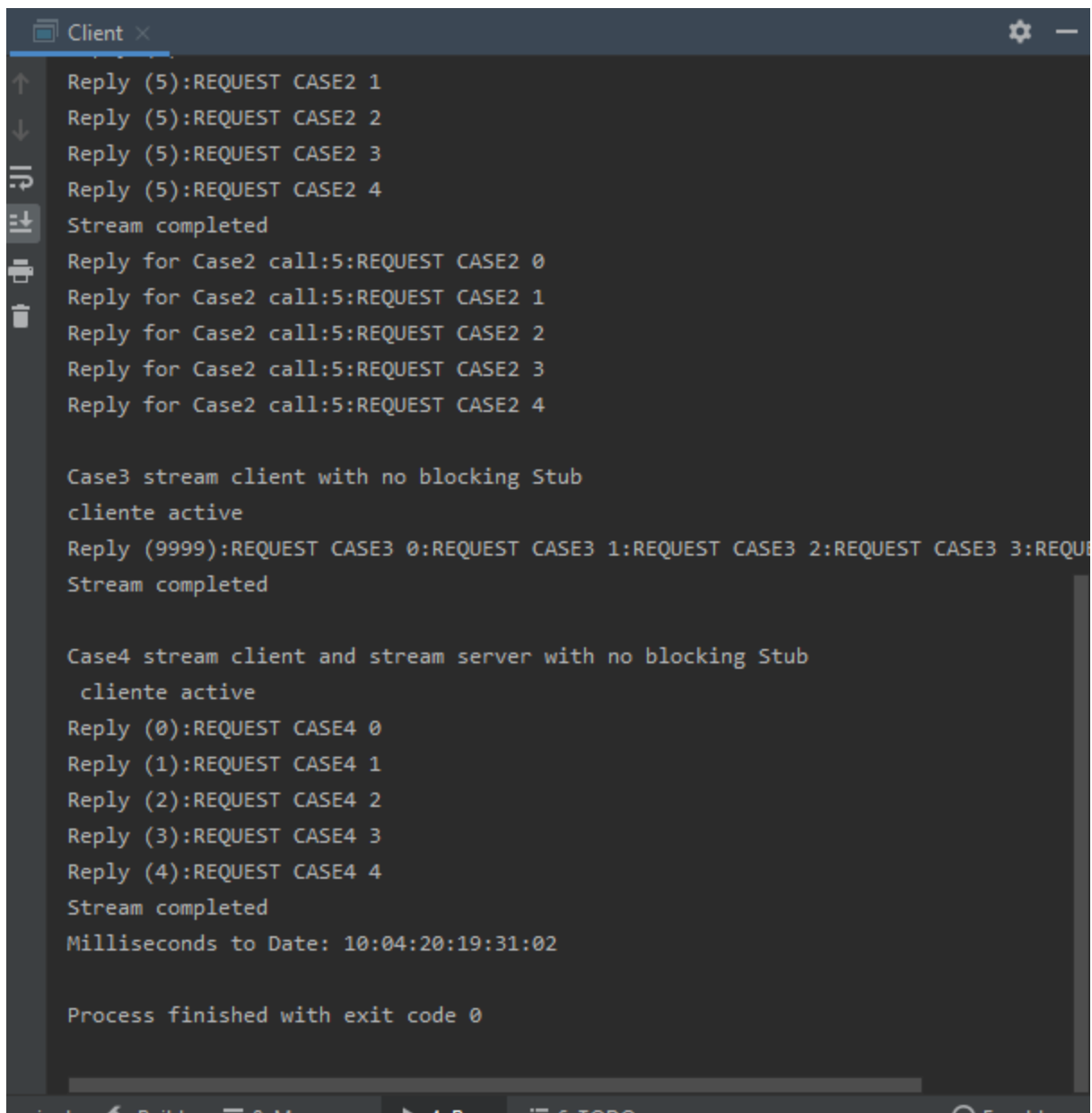


The screenshot shows the IntelliJ IDEA interface with a tab for 'Client.java' and a 'Maven' tab. The console window displays the following output:

```
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\jbr\bin\jav
Server is alive

Case1 Unário blocking stub:
Reply(0):REQUEST 0
Reply(1):REQUEST 1
Reply(2):REQUEST 2
Case1 with no blocking Stub
    cliente active
    cliente active
    cliente active
Reply (100):REQUEST ASSINCRONO
Stream completed
Reply for Case1 call:100:REQUEST ASSINCRONO
Case1 with Future
waiting futures completed
waiting futures completed
waiting futures completed
Future reply:RES=INVOKED WITH FUTURE

Case2 stream server blocking stub:
Reply for Case2 BlockStub:5:REQUEST CASE2 0
Reply for Case2 BlockStub:5:REQUEST CASE2 1
Reply for Case2 BlockStub:5:REQUEST CASE2 2
Reply for Case2 BlockStub:5:REQUEST CASE2 3
Reply for Case2 BlockStub:5:REQUEST CASE2 4
Case2 with no blocking Stub
Active and waiting for Case2 completed
Reply (5):REQUEST CASE2 0
```



```
Client x
↑ Reply (5):REQUEST CASE2 1
↓ Reply (5):REQUEST CASE2 2
⏮ Reply (5):REQUEST CASE2 3
⏭ Reply (5):REQUEST CASE2 4
⏮ Stream completed
⏭ Reply for Case2 call:5:REQUEST CASE2 0
⏮ Reply for Case2 call:5:REQUEST CASE2 1
⏭ Reply for Case2 call:5:REQUEST CASE2 2
⏮ Reply for Case2 call:5:REQUEST CASE2 3
⏭ Reply for Case2 call:5:REQUEST CASE2 4

Case3 stream client with no blocking Stub
cliente active
Reply (9999):REQUEST CASE3 0:REQUEST CASE3 1:REQUEST CASE3 2:REQUEST CASE3 3:REQUEST CASE3 4
Stream completed

Case4 stream client and stream server with no blocking Stub
cliente active
Reply (0):REQUEST CASE4 0
Reply (1):REQUEST CASE4 1
Reply (2):REQUEST CASE4 2
Reply (3):REQUEST CASE4 3
Reply (4):REQUEST CASE4 4
Stream completed
Milliseconds to Date: 10:04:20:19:31:02

Process finished with exit code 0
```

Verificando assim o correto funcionamento dos 4 tipos de chamadas que existem através do middleware gRPC.

Exercício 3)

Alínea a) Foi solicitado a criação do próprio contrato seguindo os passos que estão no enunciado. Inicialmente o contrato tinha apenas a definição necessária para ser utilizado o método findPrimes, sendo que mais tarde foi redefinido para os métodos propostos, ficando como resultado final:

```

syntax = "proto3" ;
// each class is defined in each own file, inside a common package
option java_multiple_files = true;
option java_package = "operationsservice" ;
package operationsservice; // package do proto
// The greeting service definition.
service OperationService {
    rpc findPrimes (OperationRequest) returns (stream OperationReply); //Alínea c
    rpc sumIntNumbers (stream OperationRequest) returns (OperationReply); //Alínea e
    rpc sumStreamIntNumbers (stream OperationRequest) returns (stream OperationReply); //Alínea f
    rpc findPrimesStream (stream OperationRequest) returns (stream OperationReply); //Alínea g
}

// input message
message OperationRequest {
    int32 opRequest1 = 1 ; //Anteriormente Num
    int32 opRequest2 = 2 ; //Anteriormente Start
}

// output message
message OperationReply {
    int32 opReply = 1 ;
}

```

Alínea b) Após gerado o artefacto da alínea anterior, foi necessário alterar o ficheiro pom.xml comum às várias pastas do trabalho (cliente, servidor e contrato) nomeadamente o artifactId de cada pasta assim como o caminho específico para o contrato, para que este incluísse as novas dependências gRPC e do contrato gerado:

```

<dependencies>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-netty-shaded</artifactId>
        <version>1.28.0</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-protobuf</artifactId>
        <version>1.28.0</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-stub</artifactId>
        <version>1.28.0</version>
    </dependency>
    <!-- Dependência explícita -->
    <dependency>
        <groupId>cn</groupId>
        <artifactId>PrimesServiceImpl</artifactId>
        <version>1.0</version>
        <scope>system</scope>
        <systemPath>C:/Users/asus/Desktop/CN/Trabalhos_CN/Lab2/gRPCPrimesProject/lib/OperationServiceContract-1.0.jar</systemPath>
    </dependency>
</dependencies>

```

Alínea c)

Com o novo contrato criado e com base no Código fornecido pelos docentes, procedeu-se à implementação do código que permitia ao cliente correr o serviço pretendido. Como já realizado no exercício 1, o objetivo seria implementar a função findPrimes imprimindo um certo número de números primos dado pelo utilizador começando por um valor inicial. Assim, para implementar o cliente utilizou-

se a função `case2()` do exemplo fornecido sendo esta adequada para o exercício onde a partir de todos os números, recebeu-se uma stream dos números que eram primos (como indicado no contrato para o método `findPrimes`) para stub bloqueante e de seguida para não bloqueante do lado do servidor (Streaming servidor). O contrato terá um papel de intermediário entre o cliente o servidor, assim as classes `OperationRequest` e `OperationReply` são classes indicadas pelo Maven que irão servir como pedido e resposta na “conversação” entre o cliente e o servidor, o Maven indica também o `setOpRequest1` e `setOpRequest2` onde serão passados os parâmetros necessários (neste caso o `numOfPrimes` e o `start`). Já no caso do servidor, colocou-se as funções `isPrime` e `findPrime` implementadas no exercício 1) e colocando este á escuta. Criou-se a classe `ClientStreamObserver` novamente a partir do código fornecido, redefinindo os métodos que esta implementa para, neste caso, adicionar a cada resposta o número primo (alterando assim o método `onNext()`):

Servidor do lado esquerdo e Cliente do lado direito:

```

package Server;

import com.google.protobuf.Timestamp;
import io.grpc.ServerBuilder;
import io.grpc.stub.StreamObserver;
import primeservice.NumOfPrimes;
import primeservice.Prime;

// ...

FindPrimes stream server blocking stub:
Reply for FindPrimes BlockStub: 3
Reply for FindPrimes BlockStub: 5
Reply for FindPrimes BlockStub: 7
Reply for FindPrimes BlockStub: 11
Reply for FindPrimes BlockStub: 13
Reply for FindPrimes BlockStub: 17
Reply for FindPrimes BlockStub: 19
FindPrimes with no blocking Stub
Active and waiting for FindPrimes completed
Reply (3)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (5)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (7)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (11)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (13)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed

```

Alínea d)

Explicada no final do relatório uma vez que os testes foram feitos para todas as funções na mesma execução

Alínea e)

Foi solicitado a implementação de uma nova operação que permita enviar um stream de números inteiros, obtendo como resposta a soma de todos eles. Para esta implementação, começou-se por fazer o Cliente partindo novamente do exemplo de código fornecido pelos docentes, neste caso aproveitando principalmente a função `case3()` onde a partir de uma stream de números enviados pelo cliente, o servidor responde apenas com um único reply: O somatório de todos os números da stream (Streaming do Cliente). Assim, fez-se a função `sumIntNumbers` que recebe um conjunto de numbers e a cada número recebido, colocava o número no request através do array numbers passado na assinatura. Desta vez, criou-se uma nova classe `ClientSumStreamObserver` onde se redifiniu novamente os métodos, mais precisamente o `onNext()` para que na resposta venha o somatório dos números. Do lado do servidor, chama-se a função `sumIntNumbers` do cliente e, criando um `StreamObserverStream`, o método

onNext() fará efetivamente o somatório de todos os números, sendo que o onComplete() passa novamente ao contrato a resposta que o cliente irá ler.

Alínea f)

Neste alínea foi solicitado o acréscimo de uma nova operação onde o servidor recebe uma stream de números inteiros aos pares e retorna a soma de cada par ao cliente em forma de stream. Tendo em conta esta abordagem, recorreu-se novamente ao código fornecido, sendo que para esta situação o case4() seria o mais indicado uma vez que ambos trabalham com streams (Streaming de cliente e servidor).

Assim, no cliente criou-se a função sumStreamIntNumbers que vai enviar ao servidor um stream de números inteiros. No servidor implementa-se a função que recebe a stream passada pelo cliente que tem a string de inteiros (separada por vírgula), sendo que caso o array seja par, é feita a soma do element corrente com o seguinte sendo o resultado retornado em forma de stream, caso seja impar o ultimo elemento não tem mais nenhum com que somar então retorna-se a ele próprio.

O conceito desta alínea é semelhante á anterior sendo que neste caso como o servidor quer enviar uma resposta em stream de números para o cliente. Assim, redifiniu-se novamente os métodos onNext() de tanto do cliente como do servidor para que a cada onNext() do cliente o servidor some os pares e retorne imediatamente pelo onNext(), não esperando pelo envio dos outros pares de números.

Alínea g)

Nesta exercício mudou-se o método findPrimes em que agora este recebe um intervalo de números [inicio,fim] e é obtida uma stream com todos os primos desse intervalo.

O cliente implementa então uma função findPrimesStream que a partir de um intervalo de números de 1 a 100 divide-o em 4 intervalos. A cada intervalo chama-se o método findPrimes() em cada onNext() onde a stream termina no onComplete(). No servidor cada pedido é processado da mesma maneira que o exercício 3 c) sendo que retorna uma stream de números inteiros que se encontram no intervalo pretendido. Criou-se uma classe StreamObserverFindPrimesStream onde se redifiniu o método onNext() onde a cada onNext() do cliente são feitos vários onNext() a cada resposta do pedido. O método onComplete() termina quando nao existirem mais pedidos.

Alínea d)

Após implementadas todas as funções e testadas isoladamente, realizou-se um conjunto de testes onde se propõe a utilização de três cenários:

- 1) Cliente e servidor local;
- 2) Cliente local e servidor numa VM do GCP;
- 3) Cliente e servidor em VMs diferentes no GCP usando os IPs internos.

No cenário 1) realizou-se o teste de todas as funções numa execução entre o cliente e servidor local:

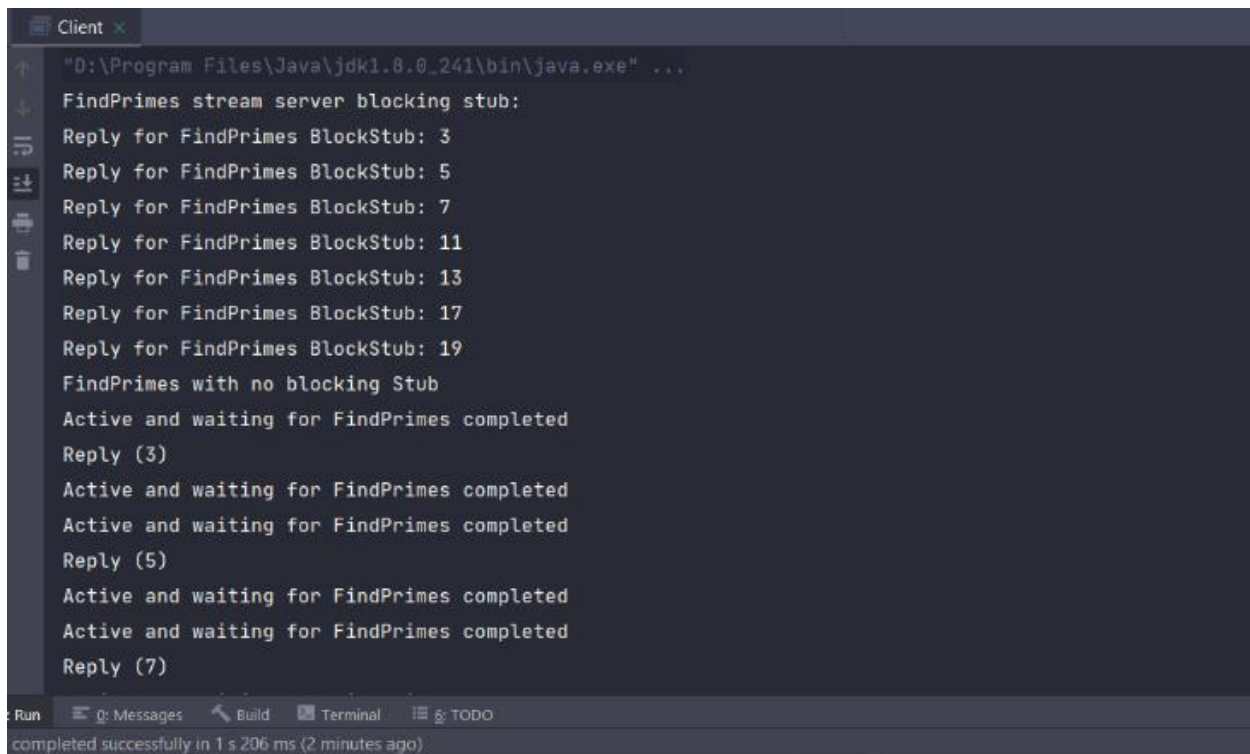
Cliente: Nos vários system.out.println colocou-se o nome das funções que correspondem a cada alínea descrita sendo:

FindPrimes – Com stub bloqueante e não bloqueante Alínea 3 c)

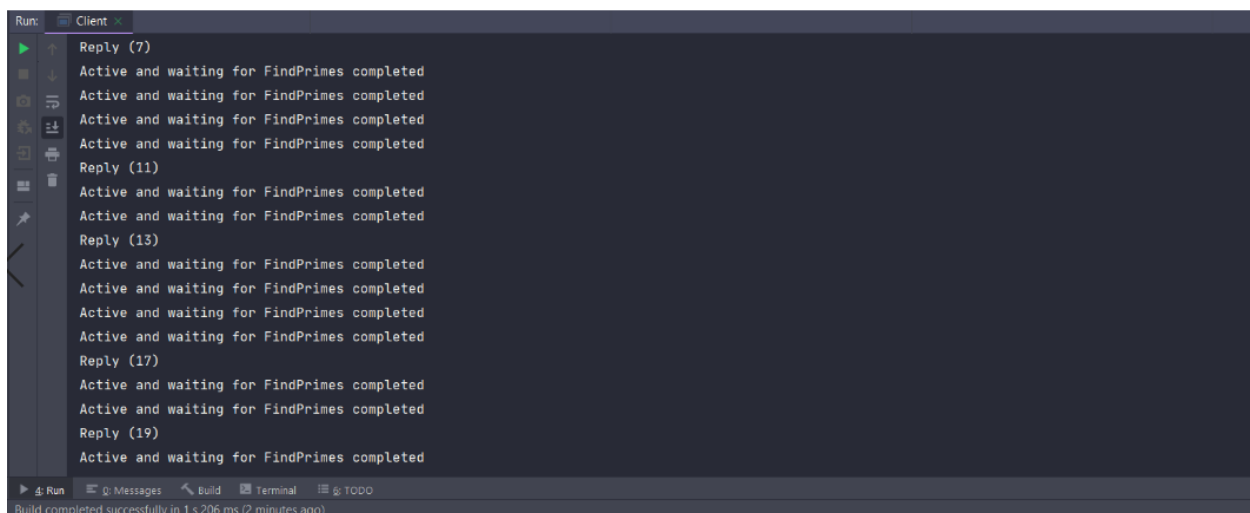
sumIntNumbers – Alínea 3 e)

sumStreamIntNumbers – Alínea 3 f)

findPrimesStream – Alínea 3 g)



```
Client x
"D:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
FindPrimes stream server blocking stub:
Reply for FindPrimes BlockStub: 3
Reply for FindPrimes BlockStub: 5
Reply for FindPrimes BlockStub: 7
Reply for FindPrimes BlockStub: 11
Reply for FindPrimes BlockStub: 13
Reply for FindPrimes BlockStub: 17
Reply for FindPrimes BlockStub: 19
FindPrimes with no blocking Stub
Active and waiting for FindPrimes completed
Reply (3)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (5)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (7)
Run Messages Build Terminal TODO
completed successfully in 1 s 206 ms (2 minutes ago)
```



```
Run Client
Reply (7)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (11)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (13)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (17)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (19)
Active and waiting for FindPrimes completed
Run Messages Build Terminal TODO
Build completed successfully in 1 s 206 ms (2 minutes ago)
```

```
Client x
Active and waiting for FindPrimes completed
Stream completed
Reply for FindPrimes call: 3
Reply for FindPrimes call: 5
Reply for FindPrimes call: 7
Reply for FindPrimes call: 11
Reply for FindPrimes call: 13
Reply for FindPrimes call: 17
Reply for FindPrimes call: 19

sumIntNumbers stream client with no blocking Stub
cliente active
Reply (400)
Stream completed

sumStreamIntNumbers stream client and stream server with no blocking Stub
cliente active
Reply (200)
```

Run Messages Build Terminal TODO
completed successfully in 1 s 206 ms (3 minutes ago)

```
Client x
Reply (200)
Reply (200)
Stream completed

sumStreamIntNumbers stream client and stream server with no blocking Stub
cliente active
Reply (200)
Reply (200)
Reply (100)
Stream completed

findPrimesStream stream client and stream server with no blocking Stub

Primeira Parte
Reply (2)
Reply (3)
Reply (5)
Reply (7)
```

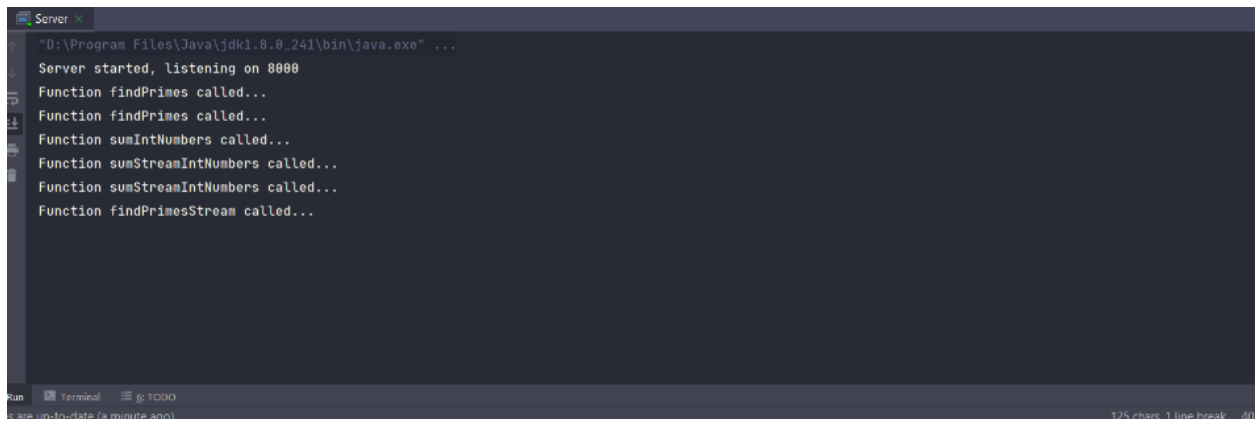
```
Run Client x
Terceira Parte
Reply (53)
Reply (59)
Reply (61)
Reply (67)
Reply (71)
Reply (73)

Quarta Parte
Reply (79)
Reply (83)
Reply (89)
Reply (97)
cliente active
Stream completed

Process finished with exit code 0
```

Run Messages Build Terminal TODO
Event Log

Servidor:

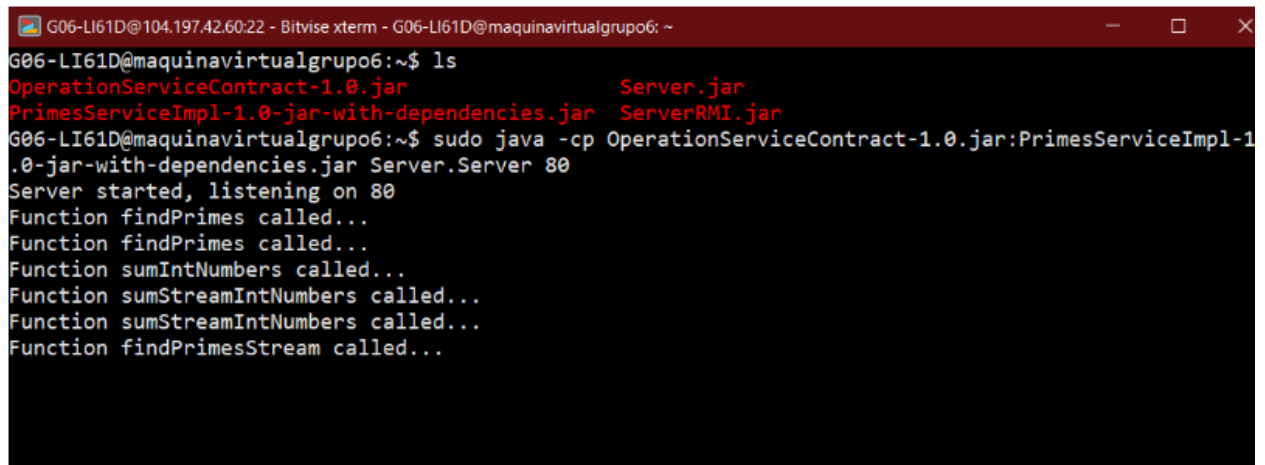


```
"D:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Server started, listening on 8000
Function findPrimes called...
Function findPrimes called...
Function sumIntNumbers called...
Function sumStreamIntNumbers called...
Function sumStreamIntNumbers called...
Function findPrimesStream called...
```

No cenário 2) Fez-se a mesma execução onde o cliente é local e o servidor numa VM:

Cliente: Para o cliente os prints são os mesmos utilizados para o cliente do cenário 1)

Servidor:



```
G06-LI61D@104.197.42.60:22 - Bitvise xterm - G06-LI61D@maquinavirtualgrupo6: ~
G06-LI61D@maquinavirtualgrupo6:~$ ls
OperationServiceContract-1.0.jar          Server.jar
PrimesServiceImpl-1.0-jar-with-dependencies.jar  ServerRMI.jar
G06-LI61D@maquinavirtualgrupo6:~$ sudo java -cp OperationServiceContract-1.0.jar:PrimesServiceImpl-1.0-jar-with-dependencies.jar Server.Server 80
Server started, listening on 80
Function findPrimes called...
Function findPrimes called...
Function sumIntNumbers called...
Function sumStreamIntNumbers called...
Function sumStreamIntNumbers called...
Function findPrimesStream called...
```

Por ultimo, resta testar o cenário 3, onde é colocado o cliente o servidor em VMs distintas:

Cliente:

```

G06-LI61D@104.197.42.60:22 - Bitwise xterm - G06-LI61D@maquinavirtualgrupo6: ~
G06-LI61D@maquinavirtualgrupo6:~$ sudo java -cp OperationServiceContract-1.0.jar:PrimesBase
Client-1.0-jar-with-dependencies.jar clientApp.Client
FindPrimes stream server blocking stub:
Reply for FindPrimes BlockStub: 3
Reply for FindPrimes BlockStub: 5
Reply for FindPrimes BlockStub: 7
Reply for FindPrimes BlockStub: 11
Reply for FindPrimes BlockStub: 13
Reply for FindPrimes BlockStub: 17
Reply for FindPrimes BlockStub: 19
FindPrimes with no blocking Stub
Active and waiting for FindPrimes completed
Reply (3)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (5)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (7)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (11)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (13)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (17)
Active and waiting for FindPrimes completed
Active and waiting for FindPrimes completed
Reply (19)
Active and waiting for FindPrimes completed
Stream completed
Reply for FindPrimes call: 3
Reply for FindPrimes call: 5
Reply for FindPrimes call: 7
Reply for FindPrimes call: 11
Reply for FindPrimes call: 13
Reply for FindPrimes call: 17
Reply for FindPrimes call: 19

sumIntNumbers stream client with no blocking Stub
cliente active
Reply (400)
Stream completed

sumStreamIntNumbers stream client and stream server with no blocking Stub
cliente active
Reply (200)
Reply (200)
Stream completed

sumStreamIntNumbers stream client and stream server with no blocking Stub
cliente active
Reply (200)
Reply (200)
Reply (100)
Stream completed

findPrimesStream stream client and stream server with no blocking Stub

Primeira Parte
Reply (2)
Reply (3)
Reply (5)
Reply (7)
Reply (11)
Reply (13)
Reply (17)
Reply (19)
Reply (23)

Segunda Parte
Reply (29)
Reply (31)
Reply (37)
Reply (41)
Reply (43)
Reply (47)

Terceira Parte

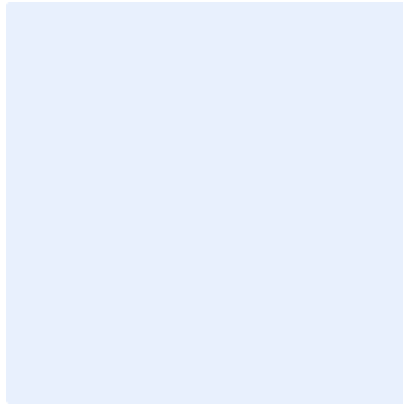
```

E Servidor:

```

G06-LI61D@104.197.42.60:22 - Bitwise xterm - G06-LI61D@maquinavirtualgrupo6: ~
G06-LI61D@maquinavirtualgrupo6:~$ ls
OperationServiceContract-1.0.jar          Server.jar
PrimesServiceImpl-1.0-jar-with-dependencies.jar  ServerRMI.jar
G06-LI61D@maquinavirtualgrupo6:~$ sudo java -cp OperationServiceContract-1.0.jar:PrimesServiceImpl-1
.0-jar-with-dependencies.jar Server.Server 80
Server started, listening on 80
Function findPrimes called...
Function findPrimes called...
Function sumIntNumbers called...
Function sumStreamIntNumbers called...
Function sumStreamIntNumbers called...
Function findPrimesStream called...

```



4. Resumo dos problemas encontrados e as soluções aplicadas:

- ➔ Inicialmente, o contrato foi alterado várias vezes á medida que se iam implementando as novas funções, onde se estava a criar uma message de input e de output com os parâmetros necessários para cada função. Mais tarde percebeu-se que seria possível a utilização de um OperationRequest e OperationReply genérico assim como todos os seus gets() e sets() que podia ser utilizado em todas as funções, não criando assim código redundante.
- ➔ Outro problema encontrado for na geração do ficheiro jar onde tivemos alguns problemas ao fazer package do Maven, onde aparecia que o “caminho” do contrato não existia sendo que se estava a observar o caminho correto do ficheiro dentro da pasta, ao mudar a pasta para o ambiente trabalho

5. Indicação se a solução final é executável e demonstrável

As soluções para as funções implementadas são executáveis

6. Conclusões e lições aprendidas

Com este trabalho aprendeu-se a trabalhar com mecanismos de chamada remota. Inicialmente estudou-se o serviço RMI e percebeu-se as limitações das abordagem no uso de callbacks, problemas em traduções de ips devido a protocolos externos etc...

Já na abordagem do gRPC notou-se uma evolução em relação ao serviço anterior não encontrando problemas que foram identificados na resolução do primeiro exercício. Aprendeu-se a analisar ficheiros como o pom.xml , explorou-se o Maven e as suas capacidades.

7. Auto-avaliação qualitativa por parte dos alunos

Bom