Laboratório 2

Objectivos:

- 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

Pretende-se desenvolver serviços para cálculo de números primos, usando RMI e gRPC. 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.

1) Desenvolva o serviço usando a tecnologia RMI. Considere o contrato seguinte, composto por duas interfaces: `IPrimesService` e `ICallback`. Na interface `IPrimesService` o método `findPrimes` recebe o valor inicial a partir do qual se irá procurar números primos, a quantidade de números a obter, e o *callback* por onde as respostas serão enviadas.

```
public interface IPrimesService extends Remote {
   void findPrimes(int start,
                   int numberOfPrimes,
                   ICallback listener) throws RemoteException;
}

public interface ICallback extends Remote {
   void nextPrime(int prime) throws RemoteException;
}
```

Contrato RMI do serviço de geração de números primos

   a) Implemente o servidor RMI. Note que a implementação da operação `findPrimes` não deve bloquear a chamada do cliente, retornando os primos através do *callback* passado na chamada.

```
static boolean isPrime(int num) {
  if (num <= 1) return false;
  if (num == 2 || num == 3) return true;
  if (num % 2 == 0) return false;
  for (int i=3; i <= Math.sqrt(num); i+=2) {
     if (num % i == 0) return false;
  }
  return true;
}
```

Sugestão de uma função para determinar se um número é primo

   b) Desenvolva o cliente do serviço, o qual lê do *standard input* o número inicial (`start`) e a quantidade de primos desejados (`numberOfPrimes`), e mostra no *standard output* a sequência de primos retornados pelo servidor.

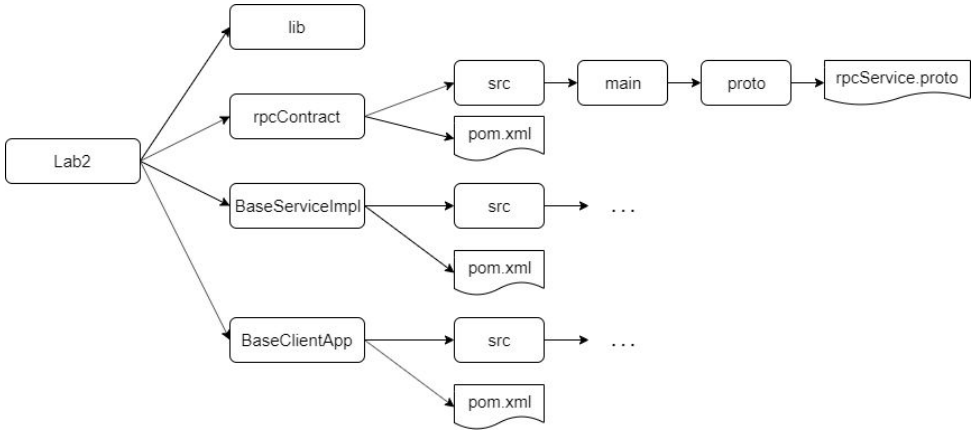
c) Proponha e implemente alterações à solução (contrato, cliente ou servidor) para que seja possível a um cliente realizar 2 ou mais chamadas ao serviço e distinguir as respostas de cada chamada.

d) Realize testes de execução considerando três cenários: i) cliente e servidor local ii) cliente local e servidor numa VM do GCP iii) cliente e servidor em VMs diferentes no GCP usando os IPs internos das VM.

2) Em anexo é fornecido o exemplo `gRPCBaseProjectCompleto` que foi apresentado na aula teórica e documentado nos slides. Verifique a estrutura dos vários projetos *Maven* em *IntelliJ*, como apresentado na Figura 1.a). **Em anexo apresenta-se como abrir os projetos Maven no *Intellij***.

Sempre que fizer alterações ao código do contrato (projeto `rpcContract`) deve fazer duplo *click* na opção *package* do *Lifecycle* Maven, como indicado na Figura 1.b). Note que o projeto `rpcContract` gera um artefacto em `target\rpcContract-1.0.jar` que tem de ser posteriormente copiado para a diretoria `lib`.

Modificando, eventualmente, o código da aplicação cliente (`BaseClientApp`), realize chamadas ao servidor para as diversas operações, analisando os diferentes padrões de chamada.



a) b)

Fig. 1: a) Estrutura de directorias e ficheiros dos projetos fornecidos; b) Janela Maven no IntelliJ

Note que para lançar o servidor na linha de comando, à escuta no porto 8000, deve indicar o local (`-cp classpath`) onde estão as classes do servidor e do contrato e indicar a *main class*:

`java -cp rpcContract-1.0.jar;BaseServiceImpl-1.0-jar-with-dependencies.jar serverapp.Server 8000`

`(em linux o separador dos Jar é o caractere : )`
`(se usar o porto 80 não se esqueça que tem de executar o comando com` **`sudo`**`)`

3) Pretende-se implementar o serviço de números primos em tecnologia gRPC, cujo contrato é apresentado em seguida.

   Deve seguir a mesma estrutura de diretorias da Figura 1a), com as devidas adaptações de nomes (ou seja, diretoria `lib` e projectos independentes para contrato, servidor e cliente).

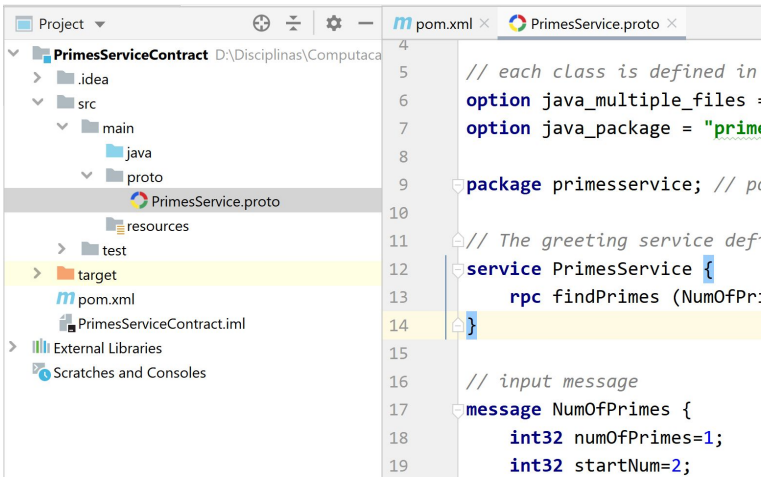
```
syntax = "proto3";

// each class is defined in each own file, inside a common package
option java_multiple_files = true;
option java_package = "primesservice";

package primesservice; // package do proto

// The greeting service definition.
service PrimesService {
   rpc findPrimes (NumOfPrimes) returns (stream Prime);
}

// input message
message NumOfPrimes {
   int32 numOfPrimes = 1;
   int32 startNum = 2;
}

// output message
message Prime {
   int32 prime = 1;
}
```

Contrato protobuf do serviço de geração de números primos

a) Defina o contrato *protobuf* num novo projeto Maven no IntelliJ. Note que a criação do ficheiro `.proto` é feita através da criação de um ficheiro geral (*new -> File*), dentro da diretoria `src\main\proto`.

   A figura apresenta um excerto do resultado esperado.

   Adapte o `pom.xml` para incluir as dependências que foram usadas no contrato da alínea 2.



   Gere o JAR com o contrato e coloque-o na diretoria `lib`, exterior ao projeto.

b) Implemente o serviço `PrimesService`, o qual depende do artefato gerado na alínea anterior. Crie um novo projeto *Maven* no *IntelliJ* e adapte o `pom.xml` para incluir as dependências gRPC e a dependência do contrato gerado anteriormente (à semelhante do projeto do servidor na alínea 2).

c) Implemente um cliente que acede ao serviço com *stubs*: i) bloqueantes e ii) não bloqueantes

   Deve criar um novo projeto Maven no IntelliJ e realizar as alterações necessárias ao `pom.xml` para incluir as dependências gRPC e do contrato gerado na alínea a).

d) Realize testes de execução considerando três cenários: i) cliente e servidor local ii) cliente local e servidor numa VM do GCP iii) cliente e servidor em VMs diferentes no GCP usando os IPs internos.

e) Acrescente uma nova operação ao serviço que permita enviar um *stream* de números inteiros e obter como resposta a soma de todos os inteiros. Acrescente ao cliente a chamada à nova operação.

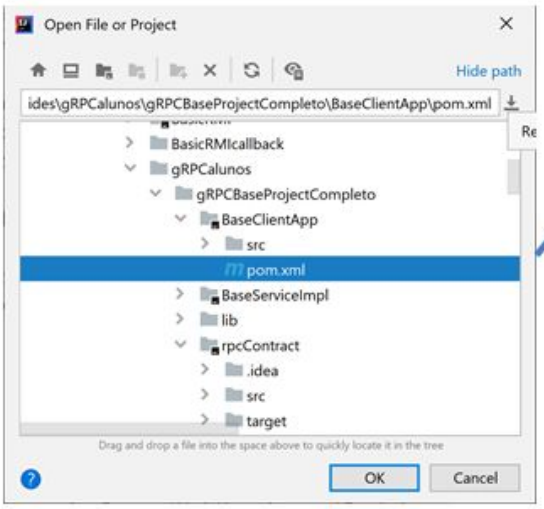
f) Acrescente uma nova operação ao serviço que permita enviar um *stream* de operações de adição (*message OperationRequest*) e obter um *stream* onde o serviço escreve o resultado de cada adição (*message OperationReply*). Acrescente ao cliente a chamada à nova operação.

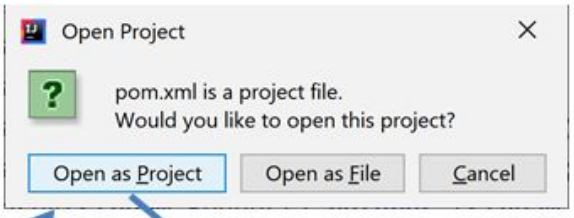
g) A operação *findPrimes*, permite obter N números primos (`numOfPrimes`) a partir de um número inicial (*startNum*). Acrescente ao serviço uma nova operação que retorna um stream de números primos no intervalo [n_1, n_2]. Acrescente ao cliente a chamada à nova operação, obtendo em paralelo os número primos do intervalo [1, 100], dividindo o intervalo em 4 partes.

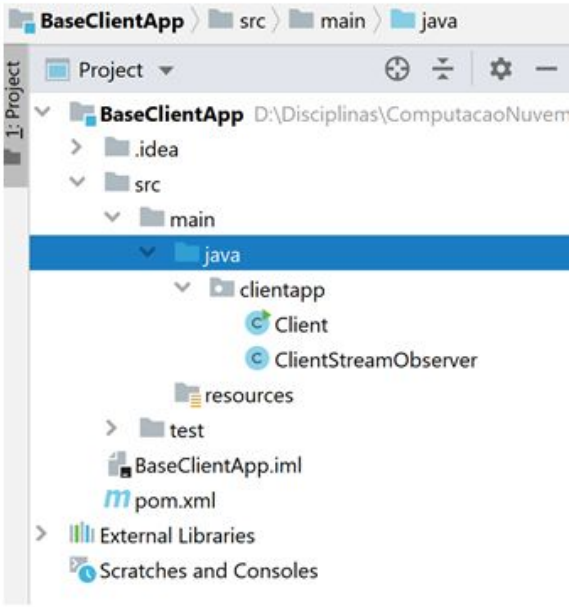
ANEXO

Como abrir projetos *Intellij* vindos de outra fonte.

Abrir o projeto selecionando na estrutura de diretorias o ficheiro pom.xml







Se usar a opção *Import project* na janela inicial do Intellij, selecione na mesma o ficheiro pom.xml