Guía paso a paso: proyecto grpc-basico (servidor gRPC con 4 tipos de streaming + cliente)
==========

1. Crear esqueleto del proyecto
```bash
mkdir grpc-basico && cd grpc-basico
cat > pom.xml <<'EOF'
# (en el siguiente paso pega el contenido completo del pom)
EOF
```

2. Sustituir el contenido de pom.xml (dependencias gRPC/protobuf y plugin de compilación de .proto)
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>com.grpc</groupId>
 <artifactId>ej1</artifactId>
 <version>1.0</version>

 <dependencies>
  <dependency>
   <groupId>com.google.protobuf</groupId>
   <artifactId>protobuf-java</artifactId>
   <version>3.22.2</version>
  </dependency>
  <dependency>
   <groupId>io.grpc</groupId>
   <artifactId>grpc-netty-shaded</artifactId>
   <version>1.54.0</version>
  </dependency>
  <dependency>
   <groupId>io.grpc</groupId>
   <artifactId>grpc-protobuf</artifactId>
   <version>1.54.0</version>
```

```
 </dependency>
 <dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-stub</artifactId>
  <version>1.54.0</version>
 </dependency>
 <dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.3.2</version>
 </dependency>
 <dependency>
  <groupId>io.reactivex.rxjava3</groupId>
  <artifactId>rxjava</artifactId>
  <version>3.1.8</version>
 </dependency>
</dependencies>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<build>
 <defaultGoal>clean generate-sources compile install</defaultGoal>
 <plugins>
  <plugin>
   <groupId>com.github.os72</groupId>
   <artifactId>protoc-jar-maven-plugin</artifactId>
   <version>3.6.0.1</version>
   <executions>
    <execution>
     <phase>generate-sources</phase>
     <goals>
      <goal>run</goal>
     </goals>
     <configuration>
      <includeMavenTypes>direct</includeMavenTypes>
      <inputDirectories>
       <include>src/main/resources</include>
      </inputDirectories>
```

```
        <!-- Descomentar SOLO SI TIENES macOS con arquitectura Intel (x86_64) --
>
        <!-- <protocArtifact>com.google.protobuf:protoc:3.22.2:exe:osx-
x86_64</protocArtifact> -->
        <outputTargets>
         <outputTarget>
          <type>java</type>
          <outputDirectory>src/main/java</outputDirectory>
         </outputTarget>
         <outputTarget>
          <type>grpc-java</type>
          <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.54.0</pluginArtifact>
          <outputDirectory>src/main/java</outputDirectory>
         </outputTarget>
        </outputTargets>
       </configuration>
      </execution>
     </executions>
    </plugin>
    <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.8.0</version>
     <configuration>
      <source>1.8</source>
      <target>1.8</target>
     </configuration>
    </plugin>
   </plugins>
  </build>
</project>
```

3. Crear la estructura de directorios base
```bash
mkdir -p src/main/resources
mkdir -p src/main/java/server
mkdir -p src/main/java/service
mkdir -p src/main/java/client
```

4. Definir el contrato gRPC en src/main/resources/hello.proto

```proto
syntax = "proto3";

option java_multiple_files = true;
option java_package = "com.grpc.ej1";
option java_outer_classname = "HelloProto";

service HelloService {

 // UNARY
 rpc unaryHello (HelloRequest) returns (HelloResponse);

 // SERVER STREAMING
 rpc serverStreamHello (HelloRequest) returns (stream HelloResponse);

 // CLIENT STREAMING
 rpc clientStreamHello (stream HelloRequest) returns (HelloResponse);

 // BIDIRECTIONAL STREAMING
 rpc bidirectionalHello (stream HelloRequest) returns (stream HelloResponse);
}

message HelloRequest {
 string name = 1;
}

message HelloResponse {
 string message = 1;
}
```

5. Generar el código Java a partir del .proto

```bash
mvn clean compile
```

El plugin protoc-jar-maven-plugin genera las clases gRPC en src/main/java/com/grpc/ej1 (HelloRequest, HelloResponse, HelloServiceGrpc, etc.).

6. Implementar el servicio gRPC con los 4 modos en
src/main/java/service/HelloService.java
```java
package service;

import com.grpc.ej1.HelloRequest;
import com.grpc.ej1.HelloResponse;
import com.grpc.ej1.HelloServiceGrpc;
import io.grpc.stub.StreamObserver;

public class HelloService extends HelloServiceGrpc.HelloServiceImplBase {

  // ----------------------------
  // 1) UNARY
  // ----------------------------
  @Override
  public void unaryHello(HelloRequest request, StreamObserver<HelloResponse>
responseObserver) {

    System.out.println("[Unary] Recibido mensaje de: " + request.getName());

    String msg = "Hola " + request.getName();
    HelloResponse resp = HelloResponse.newBuilder().setMessage(msg).build();

    System.out.println("[Unary] Enviado mensaje -> " + msg);

    responseObserver.onNext(resp);
    responseObserver.onCompleted();
  }


  // ----------------------------
  // 2) SERVER STREAMING
  // ----------------------------
  @Override
  public void serverStreamHello(HelloRequest request,
StreamObserver<HelloResponse> responseObserver) {

    System.out.println("[ServerStreaming] Recibido mensaje de: " +
request.getName());
```

```
        String name = request.getName();

        String[] mensajes = {
            "Hola " + name + ", bienvenido!",
            "Este es un ejemplo de Server Streaming.",
            "El servidor envía varios mensajes.",
            "Fin del streaming."
        };

        try {
          for (String m : mensajes) {
            HelloResponse resp = HelloResponse.newBuilder().setMessage(m).build();
            responseObserver.onNext(resp);
            System.out.println("[ServerStreaming] Enviado mensaje -> " + m);
            Thread.sleep(400);
          }
        } catch (InterruptedException e) {
          e.printStackTrace();
        }

        responseObserver.onCompleted();
    }


    // ----------------------------
    // 3) CLIENT STREAMING
    // ----------------------------
    @Override
    public StreamObserver<HelloRequest>
clientStreamHello(StreamObserver<HelloResponse> responseObserver) {

        return new StreamObserver<HelloRequest>() {

          StringBuilder builder = new StringBuilder();

          @Override
          public void onNext(HelloRequest req) {
            System.out.println("[ClientStreaming] Recibido mesnsaje de: " +
req.getName());
            if (builder.length() > 0) builder.append(", ");
            builder.append(req.getName());
```

```
        }

        @Override
        public void onError(Throwable t) {
          System.err.println("Error en ClientStreamHello: " + t.getMessage());
        }

        @Override
        public void onCompleted() {
          String msgFinal = "Saludos a: " + builder.toString();
          System.out.println("[ClientStreaming] Enviando respuesta final: " +
msgFinal);

          HelloResponse resp = HelloResponse.newBuilder()
              .setMessage(msgFinal)
              .build();
          responseObserver.onNext(resp);
          responseObserver.onCompleted();
        }
      };
    }


    // ----------------------------
    // 4) BIDIRECTIONAL STREAMING
    // ----------------------------
    @Override
    public StreamObserver<HelloRequest>
bidirectionalHello(StreamObserver<HelloResponse> responseObserver) {

      return new StreamObserver<HelloRequest>() {

        @Override
        public void onNext(HelloRequest req) {

          System.out.println("[BidirectionalStreaming] Recibido mensaje: " +
req.getName());

          String msg = "Recibido mensaje: " + req.getName();
          HelloResponse resp = HelloResponse.newBuilder()
              .setMessage(msg)
```

```
                .build();

            System.out.println("[BidirectionalStreaming] Enviando mensaje -> " +
msg);

            responseObserver.onNext(resp);
          }

          @Override
          public void onError(Throwable t) {
            System.err.println("Error en BidirectionalHello: " + t.getMessage());
          }

          @Override
          public void onCompleted() {
            responseObserver.onCompleted();
          }
        };
    }
}
```

7. Implementar el servidor gRPC en src/main/java/server/HelloServer.java
```java
package server;

import service.HelloService;
import io.grpc.Server;
import io.grpc.ServerBuilder;

public class HelloServer {

    public static void main(String[] args) throws Exception {
        Server server = ServerBuilder
            .forPort(50051)
            .addService(new HelloService())
            .build();

        System.out.println("Servidor gRPC iniciado en puerto 50051...");
        server.start();
```

```
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            System.out.println("Apagando servidor gRPC");
            server.shutdown();
        }));

        server.awaitTermination();
    }
}
```

8. Implementar el cliente gRPC en src/main/java/client/HelloClient.java

```java
package client;

import com.grpc.ej1.*;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;

import java.util.Iterator;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

public class HelloClient {

    private final ManagedChannel channel;
    private final HelloServiceGrpc.HelloServiceBlockingStub blockingStub;
    private final HelloServiceGrpc.HelloServiceStub asyncStub;

    public HelloClient(String host, int port) {
        channel = ManagedChannelBuilder.forAddress(host, port)
                .usePlaintext()
                .build();

        blockingStub = HelloServiceGrpc.newBlockingStub(channel);
        asyncStub = HelloServiceGrpc.newStub(channel);
    }

    // ----------------------
    // UNARY
    // ----------------------
```

```
  public void testUnary() {
    String msg = "Hugo";
    System.out.println("[Client] Enviando mensaje: " + msg);
    HelloRequest req = HelloRequest.newBuilder().setName(msg).build();
    HelloResponse resp = blockingStub.unaryHello(req);
    System.out.println("[Unary] Respuesta del servidor -> " + resp.getMessage());
  }

  // ----------------------
  // SERVER STREAMING
  // ----------------------
  public void testServerStreaming() {
    String msg = "Hugo";
    System.out.println("[Client] Enviando mensaje: " + msg);
    HelloRequest req = HelloRequest.newBuilder().setName(msg).build();
    Iterator<HelloResponse> it = blockingStub.serverStreamHello(req);

    System.out.println("[ServerStreaming] Respuestas del servidor ->");
    while (it.hasNext()) {
      System.out.println(" -> " + it.next().getMessage());
    }
  }

  // ----------------------
  // CLIENT STREAMING
  // ----------------------
  public void testClientStreaming() throws InterruptedException {

    CountDownLatch latch = new CountDownLatch(1);

    StreamObserver<HelloResponse> responseObserver = new
StreamObserver<HelloResponse>() {
      @Override
      public void onNext(HelloResponse value) {
        System.out.println("[ClientStreaming] Respuesta del servidor -> " +
value.getMessage());
      }

      @Override
      public void onError(Throwable t) {
        latch.countDown();
```

```
        }

        @Override
        public void onCompleted() {
          latch.countDown();
        }
      };

      String msg1 = "Luis";
      String msg2 = "Carlos";
      String msg3 = "María";
      System.out.println("[Client] Enviando mensajes: " + msg1 + ", " + msg2 + ", " +
msg3);

      StreamObserver<HelloRequest> requestObserver =
          asyncStub.clientStreamHello(responseObserver);

      requestObserver.onNext(HelloRequest.newBuilder().setName(msg1).build());
      requestObserver.onNext(HelloRequest.newBuilder().setName(msg2).build());
      requestObserver.onNext(HelloRequest.newBuilder().setName(msg3).build());

      requestObserver.onCompleted();

      latch.await();
    }

    // ----------------------
    // BIDIRECTIONAL STREAMING
    // ----------------------
    public void testBidirectionalStreaming() throws InterruptedException {

      CountDownLatch latch = new CountDownLatch(1);

      StreamObserver<HelloResponse> responseObserver = new
StreamObserver<HelloResponse>() {
        @Override
        public void onNext(HelloResponse value) {
          System.out.println("[BidirectionalStreaming] Respuesta del servidor -> " +
value.getMessage());
        }
```

```
      @Override
      public void onError(Throwable t) {
        latch.countDown();
      }

      @Override
      public void onCompleted() {
        latch.countDown();
      }
    };

    StreamObserver<HelloRequest> requestObserver =
        asyncStub.bidirectionalHello(responseObserver);

    String[] mensajes = {"Hola", "¿Qué tal?", "Probando bidirecional", "Adiós"};

    System.out.println("[Client] Enviando mensajes: " + String.join(", ", mensajes));

    for (String m : mensajes) {
      requestObserver.onNext(HelloRequest.newBuilder().setName(m).build());
      Thread.sleep(300);
    }

    requestObserver.onCompleted();
    latch.await();
  }

  public void shutdown() throws InterruptedException {
    channel.shutdown().awaitTermination(3, TimeUnit.SECONDS);
  }

  public static void main(String[] args) throws Exception {
    HelloClient client = new HelloClient("localhost", 50051);

    client.testUnary();
    client.testServerStreaming();
    client.testClientStreaming();
    client.testBidirectionalStreaming();

    client.shutdown();
  }
```

}
```

9. Compilar y generar artefacto
```bash
mvn clean package
```

10. Lanzar los procesos (en dos terminales)
```bash
# Terminal 1: servidor gRPC (puerto 50051)
mvn exec:java -Dexec.mainClass=server.HelloServer

# Terminal 2: cliente gRPC (ejecuta las 4 pruebas)
mvn exec:java -Dexec.mainClass=client.HelloClient
```

11. Flujo de prueba
- El servidor HelloServer expone los cuatro modos gRPC: unary, server streaming, client streaming, bidirectional streaming.
- El cliente HelloClient llama a cada uno en orden y muestra las respuestas por consola.
- Los stubs y mensajes se generan automáticamente al compilar desde hello.proto.
```