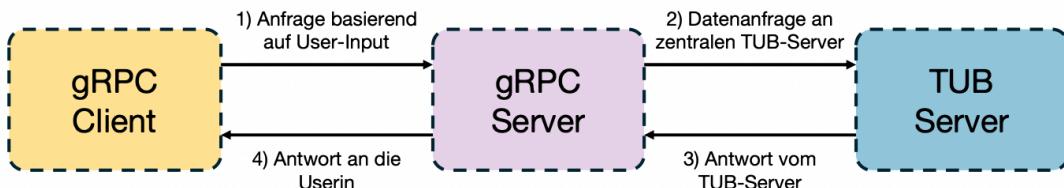


Anwendungssystemen SoSe25

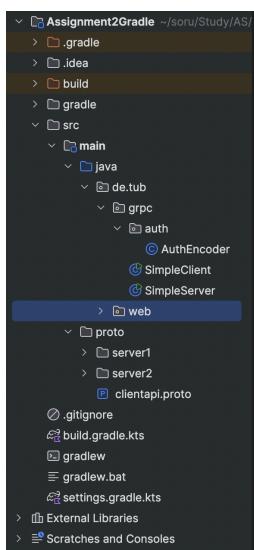
DOKUMENTATION Hausaufgabe 2

Teil 1

Aufbau:



Ordnerstruktur:



Das Projekt besteht aus gRPC-Client, lokalem Server und externem TUB-Server. Die Kommunikation erfolgt über clientapi.proto (Client–Server) und books.proto (Server–TUB). Client und Server liegen in de.tub.grpc, Authentifizierung über AuthEncoder.java. Build mit Gradle und protobuf-Plugin.

Clientapi.proto:

```
syntax = "proto3";
package clientapi;
option java_multiple_files = true;
option java_package = "clientapi";

service ClientBookService {
    rpc GetBookInfo (BookRequest) returns (BookInfoResponse);
    rpc GetAllBooks (AllBooksRequest) returns (AllBooksResponse);
    rpc GetTopBooks (TopBooksRequest) returns (TopBooksResponse);
}

message BookRequest {
    string title = 1;
}

message BookInfoResponse {
    string title = 1;
    string author = 2;
    int32 year = 3;
    int32 pages = 4;
    float average_rating = 5;
}

message AllBooksRequest {}

message AllBooksResponse {
    repeated string bookTitles = 1;
}

message TopBooksRequest {
    int32 count = 1;
}

message TopBooksResponse {
    repeated BookInfoResponse books = 1;
}
```

clientapi.proto definiert die gRPC-Schnittstelle zwischen Client und Server mit drei RPCs: GetBookInfo, GetAllBooks und GetTopBooks. Messages enthalten grundlegende Buchinformationen.

SimpleClient-Klasse:

```
public class SimpleClient {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        ManagedChannel channel = Grpc.newChannelBuilder(target: "localhost:50051", InsecureChannelCredentials.create())
            .build();

        ClientBookServiceGrpc.ClientBookServiceBlockingStub stub =
            ClientBookServiceGrpc.newBlockingStub(channel);

        boolean running = true;

        while (running) {
            System.out.println("Bitte wählen Sie eine Option:");
            System.out.println("1 - Alle verfügbaren Bücher anzeigen");
            System.out.println("2 - Informationen über ein bestimmtes Buch anzeigen");
            System.out.println("3 - Information über Top-Bücher anzeigen");
            System.out.println("4 - Beenden");
            System.out.print("Ihre Auswahl: ");

            String input = scanner.nextLine();

            switch (input) {
                case "1":
                    try {
                        AllBooksResponse response = stub.getAllBooks(AllBooksRequest.newBuilder().build());
                        List<String> bookTitles = response.getBookTitlesList();

                        if (bookTitles.isEmpty()) {
                            System.out.println("Keine Bücher gefunden.");
                        } else {
                            System.out.println("Verfügbare Bücher:");
                            for (String title : bookTitles) {
                                System.out.println(" - " + title);
                            }
                        }
                    } catch (Exception e) {
                        System.err.println("Fehler beim Abrufen der Bücher: " + e.getMessage());
                    }
                    break;

                case "2":
                    System.out.print("Geben Sie den Buchtitel ein: ");
                    String title = scanner.nextLine();

                    try {
                        BookRequest request = BookRequest.newBuilder().setTitle(title).build();
                        BookInfoResponse response = stub.getBookInfo(request);

                        System.out.println("Buchinformationen:");
                        System.out.println("Titel: " + response.getTitle());
                        System.out.println("Autor: " + response.getAuthor());
                        System.out.println("Jahr: " + response.getYear());
                        System.out.println("Seiten: " + response.getPages());
                        System.out.println("Durchschnittliche Bewertung: " + response.getAverageRating());
                    } catch (Exception e) {
                        System.err.println("Fehler beim Abrufen der Buchinformation: " + e.getMessage());
                    }
                    break;

                case "4":
                    running = false;
                    System.out.println("Client wird beendet.");
                    break;
                case "3":
                    try {
                        System.out.print("Wie viele Top-Bücher möchten Sie sehen? (z.B 3): ");
                        String countStr = scanner.nextLine();
                        int count = Integer.parseInt(countStr);

                        TopBooksRequest topReq = TopBooksRequest.newBuilder()
                            .setCount(count)
                            .build();

                        TopBooksResponse topResp = stub.getTopBooks(topReq);

                        List<BookInfoResponse> topBooks = topResp.getBooksList();

                        if (topBooks.isEmpty()) {
                            System.out.println("Keine bewerteten Bücher gefunden.");
                        } else {
                            System.out.println("Top " + count + " Bücher nach Bewertung:");
                            for (int i = 0; i < topBooks.size(); i++) {
                                BookInfoResponse b = topBooks.get(i);
                                System.out.printf("%d. %s von %s (%.2f Sterne)\n",
                                    i + 1,
                                    b.getTitle(),
                                    b.getAuthor(),
                                    b.getAverageRating());
                            }
                        }
                    } catch (Exception e) {
                        System.err.println("Fehler beim Abrufen der Top-Bücher: " + e.getMessage());
                    }
                    break;
            }
        }
    }
}
```

SimpleClient ist eine Java-Konsolenanwendung mit Menü, die per gRPC Buchlisten, Buchdetails und Top-Bewertungen vom lokalen Server abruft. Eingaben erfolgen über Scanner.

Die Konsolenanwendung bietet folgende Funktionen:

- 1.Alle verfügbaren Buchtitel anzeigen
- 2.Details zu einem bestimmten Buch abrufen
- 3.Top-N Bücher nach Bewertung anzeigen
- 4.Anwendung beenden

Books.proto:

```
service BookRating {
    rpc RateBook (RatingRequest) returns (RatingResponse);
    rpc GetAllBooks (AllBooksRequest) returns (AllBooksResponse);
    rpc GetBookInfo (BookInfoRequest) returns (BookInfoResponse);
}

message RatingRequest {
    string auth = 1;
    string bookTitle = 2;
    int32 rating = 3; // valid values 1-10
}

message RatingResponse {
    string bookTitle = 1;
    float averageRating = 2;
    int32 totalRatings = 3;
}

message AllBooksRequest {
    string auth = 1;
}

message AllBooksResponse {
    repeated string bookTitles = 1;
}

message BookInfoRequest {
    string auth = 1;
    string bookTitle = 2;
}

message BookInfoResponse {
    string bookTitle = 2;
    string author = 3;
    int32 year = 4;
    int32 pages = 5;
```

Für meine Project habe ich für diese Szenario entschiedet.

Diese .proto-Datei definiert den gRPC-Service BookRating des TUB-Servers.

Sie bietet drei RPC-Methoden:

- 1.GetAllBooks – gibt alle verfügbaren Buchtitel zurück
- 2.GetBookInfo – liefert Details zu einem bestimmten Buch
- 3.RateBook – ermöglicht das Bewerten eines Buchs (1-10 Sterne)

SimpleServer-Klasse:

```
public class SimpleServer extends ClientBookServiceGrpc.ClientBookServiceBase {

    private static final String isisEmail = "taras.levankou@campus.tu-berlin.de"; 1 usage
    private static final String isisMatrikelNr = "493936"; 1 usage
    private static final String authString = AuthEncoder.generateAuthString(isisEmail, isisMatrikelNr); 4 usages
    private static final String TUB_SERVER_ADDRESS = "141.23.71.222:60002"; 3 usages
    @Override 1 usage
    public void getBookInfo(BookRequest request, StreamObserver<BookInfoResponse> responseObserver) {
        ManagedChannel tubChannel = Grpc.newChannelBuilder(TUB_SERVER_ADDRESS, InsecureChannelCredentials.create()
            .build();

        BookRatingGrpc.BookRatingBlockingStub tubStub = BookRatingGrpc.newBlockingStub(tubChannel);

        BookInfoRequest tubRequest = BookInfoRequest.newBuilder()
            .setAuth(authString)
            .setBookTitle(request.getTitle())
            .build();
        try {
            d.tub.BookInfoResponse tubResponse = tubStub.getBookInfo(tubRequest);

            BookInfoResponse clientResponse = BookInfoResponse.newBuilder()
                .setTitle(tubResponse.getBookTitle())
                .setAuthor(tubResponse.getAuthor())
                .setYear(tubResponse.getYear())
                .setPages(tubResponse.getPages())
                .setAverageRating(tubResponse.getAverageRating())
                .build();

            responseObserver.onNext(clientResponse);
            responseObserver.onCompleted();
        } catch (Exception e) {
            e.printStackTrace();
            responseObserver.onError(e);
        } finally {
            tubChannel.shutdown();
        }
    }
}
```

getBookInfo - empfängt eine Buchanfrage, holt mit Authentifizierung die Daten vom TUB-Server (gRPC) und gibt sie an den Client zurück.

```

@Override Usage
public void getAllBooks(clientapi.AllBooksRequest request,
                      StreamObserver<clientapi.AllBooksResponse> responseObserver) {
    ManagedChannel tubChannel = Grpc.newChannelBuilder(TUB_SERVER_ADDRESS, InsecureChannelCredentials.create())
        .build();

    BookRatingGrpc.BookRatingBlockingStub tubStub = BookRatingGrpc.newBlockingStub(tubChannel);

    de.tub.AllBooksRequest tubRequest = de.tub.AllBooksRequest.newBuilder()
        .setAuth(authString)
        .build();

    try {
        de.tub.AllBooksResponse tubResponse = tubStub.getAllBooks(tubRequest);

        clientapi.AllBooksResponse.Builder responseBuilder = clientapi.AllBooksResponse.newBuilder();
        responseBuilder.addAllBookTitles(tubResponse.getBookTitlesList());

        responseObserver.onNext(responseBuilder.build());
        responseObserver.onCompleted();
    } catch (Exception e) {
        e.printStackTrace();
        responseObserver.onError(e);
    } finally {
        tubChannel.shutdown();
    }
}

public void getTopBooks(clientapi.TopBooksRequest request,
                      StreamObserver<clientapi.TopBooksResponse> responseObserver) {
    ManagedChannel tubChannel = Grpc.newChannelBuilder(TUB_SERVER_ADDRESS, InsecureChannelCredentials.create())
        .build();

    BookRatingGrpc.BookRatingBlockingStub tubStub = BookRatingGrpc.newBlockingStub(tubChannel);

    try {
        de.tub.AllBooksResponse allBooksResp = tubStub.getAllBooks(
            de.tub.AllBooksRequest.newBuilder().setAuth(authString).build()
        );
        List<String> allTitles = allBooksResp.getBookTitlesList();

        List<BookInfoResponse> allBooks = new ArrayList<>();

        for (String title : allTitles) {
            de.tub.BookInfoRequest infoReq = de.tub.BookInfoRequest.newBuilder()
                .setAuth(authString)
                .setBookTitle(title)
                .build();

            de.tub.BookInfoResponse tubInfo = tubStub.getBookInfo(infoReq);

            clientapi.BookInfoResponse book = clientapi.BookInfoResponse.newBuilder()
                .setTitle(tubInfo.getBookTitle())
                .setAuthor(tubInfo.getAuthor())
                .setYear(tubInfo.getYear())
                .setPages(tubInfo.getPages())
                .setAverageRating(tubInfo.getAverageRating())
                .build();

            allBooks.add(book);
        }
        int count = request.getCount();

        List<clientapi.BookInfoResponse> topN = allBooks.stream()
            .sorted(Comparator.comparing(clientapi.BookInfoResponse::getAverageRating).reversed())
            .limit(count)
            .toList();

        clientapi.TopBooksResponse response = clientapi.TopBooksResponse.newBuilder()
            .addAllBooks(topN)
            .build();

        responseObserver.onNext(response);
        responseObserver.onCompleted();
    } catch (Exception e) {
        e.printStackTrace();
        responseObserver.onError(e);
    } finally {
        tubChannel.shutdown();
    }
}

```

```

public static void main(String[] args) throws IOException, InterruptedException {
    Server server = ServerBuilder.forPort(12345).ServerBuilder<capture of ?>
        .addService(new SimpleServer()).capture of ?
        .build();

    System.out.println("gRPC-Server läuft auf Port 12345...");
    server.start();
    server.awaitTermination();
}

```

getAllBooks empfängt eine Anfrage, ruft authentifiziert die Buchtitel vom TUB-Server ab und leitet sie direkt an den Client weiter.

getTopBooks ruft alle Bücher vom TUB-Server ab, ermittelt deren Bewertungen, sortiert sie absteigend und gibt die Top-N Bücher entsprechend der Anfrage an den Client zurück.

Die main-Methode startet den gRPC-Server auf Port 12345, registriert den Dienst SimpleServer und hält den Server aktiv.

Konsolenausgabe:

```
gRPC-Server läuft auf Port 12345...
Bitte wählen Sie eine Option:
1 - Alle verfügbaren Bücher anzeigen
2 - Informationen über ein bestimmtes Buch anzeigen
3 - Information über Top-Bücher anzeigen
4 - Beenden
Ihre Auswahl:
```

Ausgabe aus Konsole von SimpleServer und SimpleClient

Anforderungen gRPC-Client:

```
syntax = "proto3";
package clientapi;
option java_multiple_files = true;
option java_package = "clientapi";
service ClientBookService {
    rpc GetBookInfo (BookRequest) returns (BookInfoResponse);
    rpc GetAllBooks (AllBooksRequest) returns (AllBooksResponse);
    rpc GetTopBooks (TopBooksRequest) returns (TopBooksResponse);
}
message BookRequest {
    string title = 1;
}
message BookInfoResponse {
    string title = 1;
    string author = 2;
    int32 year = 3;
    int32 pages = 4;
    float average_rating = 5;
}
message AllBooksRequest {
}
message AllBooksResponse {
    repeated string bookTitles = 1;
}
message TopBooksRequest {
    int32 count = 1;
}
message TopBooksResponse {
    repeated BookInfoResponse books = 1;
}
```

1. Die Datei clientapi.proto definiert eigene Services und Messages für die Kommunikation zwischen Client und lokalem Server. Sie unterscheidet sich von der TUB-Server-Schnittstelle und enthält die RPCs GetBookInfo, GetAllBooks und GetTopBooks.

```
public class SimpleClient {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        ManagedChannel channel = Grpc.newChannelBuilder(target: "localhost:12345", InsecureChannelCredentials.create())
            .build();

        ClientBookServiceGrpc.ClientBookServiceBlockingStub stub =
            ClientBookServiceGrpc.newBlockingStub(channel);

        boolean running = true;

        while (running) {
            System.out.println("Bitte wählen Sie eine Option:");
            System.out.println("1 - Alle verfügbaren Bücher anzeigen");
            System.out.println("2 - Informationen über ein bestimmtes Buch anzeigen");
            System.out.println("3 - Information über Top-Bücher anzeigen");
            System.out.println("4 - Beenden");
            System.out.print("Ihre Auswahl: ");

            String input = scanner.nextLine();

            switch (input) {
                case "1":
                    // Bitte wählen Sie eine Option:
                    // 1 - Alle verfügbaren Bücher anzeigen
                    // 2 - Informationen über ein bestimmtes Buch anzeigen
                    // 3 - Information über Top-Bücher anzeigen
                    // 4 - Beenden
                    // Ihre Auswahl:
```

2. Der gRPC-Client wurde als Konsolenanwendung mit Menü umgesetzt. Die Benutzerin kann per Scanner Eingaben machen und folgende Aktionen ausführen: alle Bücher anzeigen, Buchdetails abrufen, Top-Bücher anzeigen oder das Programm beenden.

```

switch (input) {
    case "1":
        try {
            AllBooksResponse response = stub.getAllBooks(AllBooksRequest.newBuilder().build());
            List<String> bookTitles = response.getBookTitlesList();

            if (bookTitles.isEmpty()) {
                System.out.println(" Keine Bücher gefunden.");
            } else {
                System.out.println(" Verfügbare Bücher:");
                for (String title : bookTitles) {
                    System.out.println(" - " + title);
                }
            }
        } catch (Exception e) {
            System.err.println("Fehler beim Abrufen der Bücher: " + e.getMessage());
        }
        break;
    @Override
    public void getAllBooks(Clientapi.AllBooksRequest request,
                           StreamObserver<Clientapi.AllBooksResponse> responseObserver) {

```

```

        ManagedChannel tubChannel = Grpc.newChannelBuilder(TUB_SERVER_ADDRESS, InsecureChannelCredentials.create())
            .build();

        BookRatingGrpc.BookRatingBlockingStub tubStub = BookRatingGrpc.newBlockingStub(tubChannel);

        de.tub.AllBooksRequest tubRequest = de.tub.AllBooksRequest.newBuilder()
            .setAuth(authString)
            .build();

        try {
            de.tub.AllBooksResponse tubResponse = tubStub.getAllBooks(tubRequest);

            Clientapi.AllBooksResponse.Builder responseBuilder = Clientapi.AllBooksResponse.newBuilder();
            responseBuilder.addAllBookTitles(tubResponse.getBookTitlesList());

```

Ihre Auswahl: 1
Verfügbare Bücher:
- 1984
- The Left Hand of Darkness
- Gender Trouble
- The Dispossessed
- A Room of One's Own
- Kindred
- Algorithms to Live By
- War and Peace
- The Brothers Karamazov
- Chasing New Horizons: Inside the Epic First Mission to Pluto
- Distributed Systems: Concepts and Design
- The Republic
- Kafka on the Shore
- The Art of Computer Systems Performance Analysis

Anforderungen gRPC-Server:

```

try {
    de.tub.AllBooksResponse tubResponse = tubStub.getAllBooks(tubRequest);

    Clientapi.AllBooksResponse.Builder responseBuilder = Clientapi.AllBooksResponse.newBuilder();
    responseBuilder.addAllBookTitles(tubResponse.getBookTitlesList());

    responseObserver.onNext(responseBuilder.build());
    responseObserver.onCompleted();
}

catch (Exception e) {
    e.printStackTrace();
    responseObserver.onError(e);
} finally {
    tubChannel.shutdown();
}

try {
    de.tub.BookInfoResponse tubResponse = tubStub.getBookInfo(tubRequest);

    BookInfoResponse clientResponse = BookInfoResponse.newBuilder()
        .setTitle(tubResponse.getBookTitle())
        .setAuthor(tubResponse.getAuthor())
        .setYear(tubResponse.getYear())
        .setPages(tubResponse.getPages())
        .setAverageRating(tubResponse.getAverageRating())
        .build();

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

catch (Exception e) {
    e.printStackTrace();
    responseObserver.onError(e);
} finally {
    tubChannel.shutdown();
}

```

```

service ClientBookService {
    rpc GetBookInfo (BookRequest) returns (BookInfoResponse);
    rpc GetAllBooks (AllBooksRequest) returns (AllBooksResponse);
    rpc GetTopBooks (TopBooksRequest) returns (TopBooksResponse);
}

```

3. Der Client stellt basierend auf der Nutzereingabe gRPC-Anfragen an den lokalen Server unter Verwendung der definierten clientapi.proto. Die erhaltene Antwort wird verarbeitet (z.B. als Liste, Detailansicht oder sortiertes Ergebnis) und benutzerfreundlich in der Konsole ausgegeben.

1. Der gRPC-Server implementiert die in clientapi.proto definierten Services (GetBookInfo, GetAllBooks, GetTopBooks) und beantwortet damit die Anfragen des Clients.

Ihre Auswahl: 2
Geben Sie den Buchtitel ein: Neuromancer
Buchinformationen:
Titel: Neuromancer
Autor: William Gibson
Jahr: 1984
Seiten: 271
Durchschnittliche Bewertung: 8.8

```

public void getBookInfo(BookRequest request, StreamObserver<BookInfoResponse> responseObserver) {
    ManagedChannel tubChannel = Grpc.newChannelBuilder(TUB_SERVER_ADDRESS, InsecureChannelCredentials.create())
        .build();

    BookRatingGrpc.BookRatingBlockingStub tubStub = BookRatingGrpc.newBlockingStub(tubChannel);

    BookInfoRequest tubRequest = BookInfoRequest.newBuilder()
        .setAuth(authString)
        .setBookTitle(request.getTitle())
        .build();
}

public void getAllBooks(clientapi.AllBooksRequest request,
    StreamObserver<clientapi.AllBooksResponse> responseObserver) {

    ManagedChannel tubChannel = Grpc.newChannelBuilder(TUB_SERVER_ADDRESS, InsecureChannelCredentials.create())
        .build();

    BookRatingGrpc.BookRatingBlockingStub tubStub = BookRatingGrpc.newBlockingStub(tubChannel);

    de.tub.AllBooksRequest tubRequest = de.tub.AllBooksRequest.newBuilder()
        .setAuth(authString)
        .build();
}

```

```

int count = request.getCount();

List<clientapi.BookInfoResponse> topN = allBooks.stream()
    .sorted(Comparator.comparing(clientapi.BookInfoResponse::getAverageRating).reversed())
    .limit(count)
    .toList();

clientapi.TopBooksResponse response = clientapi.TopBooksResponse.newBuilder()
    .addAllBooks(topN)
    .build();

responseObserver.onNext(response);
responseObserver.onCompleted();

```

Ihre Auswahl: 3
 Wie viele Top-Bücher möchten Sie sehen? (z.B 3): 5
 Top 5 Bücher nach Bewertung:
 1. 1984 von George Orwell (9,68 Sterne)
 2. Networked Systems von Anirban Pathak (9,67 Sterne)
 3. Invisible Women: Data Bias in a World Designed for Men von Caroline Criado Perez (9,50 Sterne)
 4. Stone Butch Blues von Leslie Feinberg (9,22 Sterne)
 5. A Room of One's Own von Virginia Woolf (9,22 Sterne)

```

clientapi.TopBooksResponse response = clientapi.TopBooksResponse.newBuilder()
    .addAllBooks(topN)
    .build();

responseObserver.onNext(response);
responseObserver.onCompleted();

```

2. Der gRPC-Server empfängt Client-Anfragen und stellt je nach Anwendungsfall passende authentifizierte gRPC-Anfragen an den TUB-Server.

3. Im getTopBooks-Methode werden die Daten, die vom TUB-Server kommen, nicht direkt weitergeleitet, sondern zuerst verarbeitet: Es werden alle verfügbaren Bücher abgefragt, dann für jedes Buch die Detailinformationen geholt, anschließend nach Bewertung sortiert und die vom Client gewünschte Anzahl an Top-Büchern ausgewählt. Nur diese gefilterten und transformierten Daten werden schließlich an den Client zurückgesendet.

Die verarbeiteten Top-N Bücher dienen als Antwort auf die Client-Anfrage und werden als sortierte Liste zurückgegeben.

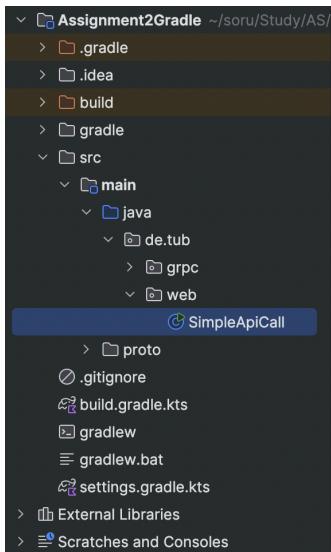
Anforderungen gRPC-Server:

Die Konsole bleibt nach der ersten Anfrage geöffnet und verarbeitet weitere Eingaben, bis die Benutzerin das Programm selbst beendet.

Der Server läuft dauerhaft und muss manuell gestoppt werden.

Teil 2

Ordnerstruktur:



Das Projekt SimpleApiCall ist eine Java-Konsolenanwendung, die mithilfe der REST Countries API die Hauptstadt und deren Koordinaten eines vom Benutzer eingegebenen Landes abruft. Diese Koordinaten werden anschließend an die Open-Meteo API übergeben, um aktuelle Wetterdaten (Temperatur und Windgeschwindigkeit) der Hauptstadt zu erhalten. Die Daten werden manuell aus den JSON-Antworten extrahiert, ohne externe Bibliotheken.

APIs:

<https://open-meteo.com/en/docs>

<https://restcountries.com/>

SimpleApiCall-Klasse:

```
public class SimpleApiCall {
    private static final HttpClient client = HttpClient.newHttpClient(); 2 usages

    public static void main(String[] args) throws IOException, InterruptedException {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.print("Geben Sie ein Land ein (oder 'exit'): ");
            String country = scanner.nextLine().trim();

            if (country.equalsIgnoreCase("exit")) {
                System.out.println("Programm beendet.");
                break;
            }

            String url = "https://restcountries.com/v3.1/name/" + country;
            HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();

            HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
            if (response.statusCode() != 200) {
                System.out.println("Land nicht gefunden.");
                continue;
            }

            String body = response.body();

            String capital = null;
            int capStart = body.indexOf("\\"capital\\":\"");
            if (capStart != -1) {
                int capEnd = body.indexOf("\\\"", capStart);
                if (capEnd != -1) {
                    capital = body.substring(capStart + 12, capEnd);
                }
            }

            Double lat = null, lon = null;
            int latStart = body.indexOf("\\"capitalInfo\\":{\\\"lat\\\":");
            if (latStart != -1) {
                int latEnd = body.indexOf("\\\"", latStart);
                if (latEnd != -1) {
                    int lonEnd = body.indexOf("\\\"", latEnd);
                    if (lonEnd != -1) {
                        try {
                            lat = Double.parseDouble(body.substring(latStart, latEnd).trim());
                            lon = Double.parseDouble(body.substring(latEnd + 1, lonEnd).trim());
                        } catch (NumberFormatException e) {
                            lat = null;
                            lon = null;
                        }
                    }
                }
            }

            if (capital == null || lat == null || lon == null) {
                System.out.println("Hauptstadt oder Koordinaten nicht gefunden.");
                continue;
            }

            System.out.println("Hauptstadt: " + capital);
            System.out.println("Koordinaten: " + lat + ", " + lon);

            String weatherUrl = "https://api.open-meteo.com/v1/forecast?latitude=" + lat +
                "&longitude=" + lon + "&current_weather=true";

            HttpRequest weatherRequest = HttpRequest.newBuilder()
                .uri(URI.create(weatherUrl))
                .GET()
                .build();
        }
    }
}
```

Beim Start zeigt das Programm ein Konsolenmenü und fordert den Benutzer auf, ein Land (auf Englisch) einzugeben. Danach ruft es die Hauptstadt sowie die geografischen Koordinaten des Landes über die REST Countries API ab und zeigt mit Hilfe der Open-Meteo API das aktuelle Wetter in der Hauptstadt an. Mit dem Befehl exit kann das Programm beendet werden. Anfrage kann man wiederholen.

Die Ausgabe erfolgt in folgender Form:

Hauptstadt: ...

Koordinaten: ...

Temperatur in Paris: ...

Windgeschwindigkeit: ...

```
HttpResponse<String> weatherResponse = client.send(weatherRequest, HttpResponse.BodyHandlers.ofString());
if (weatherResponse.statusCode() != 200) {
    System.out.println("Fehler beim Abrufen der Wetterdaten.");
    continue;
}

String weatherBody = weatherResponse.body();

String temperature = null;
String windspeed = null;

int currentStart = weatherBody.indexOf("\\"current_weather\\":");
if (currentStart != -1) {
    int tempStart = weatherBody.indexOf("\\\"temperature\\\"", currentStart);
    int windstart = weatherBody.indexOf("\\\"windspeed\\\"", currentStart);
    if (tempStart != -1) {
        int tempEnd = weatherBody.indexOf("\\\"", tempStart);
        if (tempEnd != -1) {
            temperature = weatherBody.substring(tempStart + 14, tempEnd).trim();
        }
    }
    if (windstart != -1) {
        int windEnd = weatherBody.indexOf("\\\"", windstart);
        if (windEnd != -1) {
            windspeed = weatherBody.substring(windStart + 12, windEnd).trim();
        }
    }
}

if (temperature != null && windspeed != null) {
    System.out.println("Temperatur in " + capital + ": " + temperature + " °C");
    System.out.println("Windgeschwindigkeit in " + capital + ": " + windspeed + " m/s");
}
```

Konsolenausgabe:

```
Geben Sie ein Land ein (oder 'exit'): France
Hauptstadt: Paris
Koordinaten: 48.87, 2.33
Temperatur in Paris: 33.8 °C
Windgeschwindigkeit: 7.6 km/h
-----
Geben Sie ein Land ein (oder 'exit'): exit
Programm beendet.

BUILD SUCCESSFUL in 4m 41s
```

Anforderungen:

```
String url = "https://restcountries.com/v3.1/name/" + country;
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(url))
    .GET()
    .build();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
if (response.statusCode() != 200) {
    System.out.println("Land nicht gefunden.");
    continue;
}
```



```
String weatherUrl = "https://api.open-meteo.com/v1/forecast?latitude=" + lat +
    "&longitude=" + lon + "&current_weather=true";

HttpRequest weatherRequest = HttpRequest.newBuilder()
    .uri(URI.create(weatherUrl))
    .GET()
    .build();

HttpResponse<String> weatherResponse = client.send(weatherRequest, HttpResponse.BodyHandlers.ofString());
```

```
while (true) {
    System.out.print("Geben Sie ein Land ein (oder 'exit'): ");
    String country = scanner.nextLine().trim();

    if (country.equalsIgnoreCase("exit")) {
        System.out.println("Programm beendet.");
        break;
    }
```

```
String country = scanner.nextLine().trim();

if (country.equalsIgnoreCase("exit")) {
    System.out.println("Programm beendet.");
    break;
}

String url = "https://restcountries.com/v3.1/name/" + country;
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(url))
    .GET()
    .build();
```

```
String weatherUrl = "https://api.open-meteo.com/v1/forecast?latitude=" + lat +
    "&longitude=" + lon + "&current_weather=true";
```

1. Die Anwendung nutzt REST Countries API und Open-Meteo API.

2. Durch die while-Schleife werden die APIs bei jeder Eingabe erneut aufgerufen.

3. Der Ländereingabe durch den Nutzer bestimmt die Anfrageparameter dynamisch.

```

String body = response.body();

String capital = null;
int capStart = body.indexOf("\"capital\":\"");
if (capStart != -1) {
    int capEnd = body.indexOf("]", capStart);
    if (capEnd != -1) {
        capital = body.substring(capStart + 12, capEnd);
    }
}

Double lat = null, lon = null;
int latlngStart = body.indexOf("\"capitalInfo\":{\"latlng\":[");
if (latlngStart != -1) {
    int latStart = latlngStart + "\"capitalInfo\":{\"latlng\":[\".length();
    int latEnd = body.indexOf("]", latStart);
    if (latEnd != -1) {
        int lonEnd = body.indexOf("]", latEnd);
        if (lonEnd != -1) {
            try {
                lat = Double.parseDouble(body.substring(latStart, latEnd).trim());
                lon = Double.parseDouble(body.substring(latEnd + 1, lonEnd).trim());
            } catch (NumberFormatException e) {
                lat = null;
                lon = null;
            }
        }
    }
}

```

4. Die Koordinaten aus der ersten API (Latitude und Longitude) werden aus der JSON-Antwort manuell extrahiert und anschließend als Parameter an die zweite API übergeben, um das Wetter der Hauptstadt abzufragen.

```

String weatherUrl = "https://api.open-meteo.com/v1/forecast?latitude=" + lat +
    "&longitude=" + lon + "&current_weather=true";

```

```

String weatherBody = weatherResponse.body();

String temperature = null;
String windspeed = null;

int currentStart = weatherBody.indexOf("\"current_weather\":{\"");
if (currentStart != -1) {
    int tempStart = weatherBody.indexOf("temperature\"", currentStart);
    int windStart = weatherBody.indexOf("windspeed\"", currentStart);
    if (tempStart != -1) {
        int tempEnd = weatherBody.indexOf(":", tempStart);
        if (tempEnd != -1) {
            temperature = weatherBody.substring(tempStart + 14, tempEnd).trim();
        }
    }
    if (windStart != -1) {
        int windEnd = weatherBody.indexOf(":", windStart);
        if (windEnd == -1) {
            windEnd = weatherBody.indexOf("}", windStart);
        }
        if (windEnd != -1) {
            windspeed = weatherBody.substring(windStart + 12, windEnd).trim();
        }
    }
}
if (temperature != null && windspeed != null) {
    System.out.println("Temperatur in " + capital + ": " + temperature + " °C");
}

```

