Here are the comprehensive revision notes based on the provided transcript from the class:

API Implementation and Exception Handling

Overview

In this class, we focused on the implementation of various HTTP methods within a REST API and the concepts of handling exceptions effectively using controllers in a Java-based environment. The major operations included implementing GET, POST, PUT, and DELETE requests and handling exceptions using custom exception handlers.

Key Concepts and Implementations

REST API Calls and Methods

1. **HTTP Methods in Rest Template:**
   - The class mentioned the need to implement functionality using HTTP methods like `PUT`, `PATCH`, and `DELETE` even though the `RestTemplate` does not natively support these for entities. The workaround involves using the `postForEntity` method as a template and modifying it for each required operation【8:0†transcript.txt】.
2. **GET, PUT, and POST Methods:**
   - **PUT Method:** It was highlighted that `POST` is the most similar to `PUT` due to both requiring a request body. Thus, `postForEntity` could be repurposed for `PUT` by passing a `PUT` HTTP method 【8:12†transcript.txt】.
   - **Implementation Strategy:** Copy `postForEntity` method, replace occurrences of `POST` with `PUT`, and adjust the method signature to accommodate new requirements【8:17†transcript.txt】.
3. **Handling Non-existent Methods:**

common logic in a generalized method that accepts the HTTP method as a parameter, flexibility is achieved【8:9†transcript.txt】.

Exception Handling

1. **Custom Exception Handling:**
   - The instructor emphasized creating global exception handlers. These are annotated with `@RestControllerAdvice` to handle exceptions across all controllers uniformly【8:6†transcript.txt】.
   - **Implementation**: An example is `handleExceptions` method which, upon encountering an exception, responds with an HTTP status and a message fetched from the exception itself【8:4†transcript.txt】.
2. **Using `ResponseEntity`:**
   - The use of `ResponseEntity` for returning responses was discussed, highlighting its constructors that allow setting both status codes and response bodies. It makes error handling and HTTP response customization easier【8:4†transcript.txt】【8:14†transcript.txt】.

Data Transfer Objects and Mapping

1. **Product DTO Mapping:**
   - When interfacing with external APIs, a `ProductDTO` is utilized for data representation. The class talked about creating mappers for converting between `Product` and `ProductDTO` to ensure data consistency and avoid errors due to unexpected data structures【8:14†transcript.txt】.

Practical Exercises and Common Issues

1. **Implementation Tasks:**
   - Students were tasked to implement several endpoints, including `getAllProducts`, `getProductByID`, `replaceProduct`, etc., using provided boilerplate and method signatures【8:13†transcript.txt】.
   - **Troubleshooting**: The instructor addressed common errors and encouraged students to think critically about why errors like missing class types during compile time occur, particularly emphasizing the challenges

Conclusion

The class concluded by reiterating the importance of understanding the foundational concepts of theoretical HTTP methods and practical implementation through Java's `RestTemplate`. Students were encouraged to complete the tasks to prepare for further classes, which would incorporate database integration【8:6†transcript.txt】【8:18†transcript.txt】.

This session emphasized both theoretical understanding and practical exercises to ensure that learners can navigate real-world software development challenges effectively.