Creating a series of classes is a great way to manage complexity, separate concerns, and make your code reusable. Based on the API's structure and your project's goals, here's a suggestion for the classes you should create:

**XBRLClient**

This class should be the central point of contact for the XBRL API. It's responsible for handling low-level details like API authentication, making HTTP requests, and basic error handling. This abstraction keeps the rest of your code clean and focused on business logic.

* **Attributes:**
  + base\_url: The base URL of the XBRL API.
  + api\_key or auth\_token: Your authentication credentials.
* **Methods:**
  + \_\_init\_\_(self, api\_key): Initializes the client with the provided API key.
  + \_make\_request(self, endpoint, params=None): A private helper method to handle all API calls. It constructs the full URL, adds headers, and sends the request using a library like requests. This is where you would handle potential requests.exceptions and HTTP status codes.

<br> <br>

**Company**

This class represents a specific company and its associated filings. It acts as a container for all the financial data you'll retrieve for that company.

* **Attributes:**
  + cik: The Central Index Key (CIK) of the company.
  + name: The company's name.
  + reports: A dictionary to store Report objects, keyed by fiscal year.
* **Methods:**
  + \_\_init\_\_(self, cik, name): Initializes the company with its CIK and name.
  + load\_reports(self, client, years): A method that uses the XBRLClient to query the API for 10-K report details for the specified years. For each report found, it would create a Report object and store it in the reports dictionary.

<br> <br>

**Report**

This class represents a single 10-K filing for a specific year. It serves as the bridge between the company and the detailed financial statements within that filing.

* **Attributes:**
  + report\_id: The unique API identifier for the report.
  + dts\_id: The ID of the Discoverable Taxonomy Set (DTS) associated with the report. This is critical for subsequent queries.
  + fiscal\_year: The fiscal year of the report.
  + filing\_date: The date the report was filed with the SEC.
  + statements: A dictionary to hold FinancialStatement objects, keyed by statement type (e.g., "Income Statement," "Balance Sheet").
* **Methods:**
  + \_\_init\_\_(self, report\_id, dts\_id, fiscal\_year, filing\_date): Initializes the report details.
  + load\_financial\_statements(self, client, statement\_types): A method to load specific financial statements. It would use the XBRLClient and the report's dts\_id and report\_id to query for the concepts and facts of each statement type.

<br> <br>

**FinancialStatement**

This class represents a specific financial statement (e.g., the Income Statement) and contains all the retrieved facts and concepts.

* **Attributes:**
  + statement\_type: A string identifying the type of statement (e.g., "Income Statement").
  + concepts: A list of all unique Concept objects that make up this statement.
  + facts: A list of all Fact objects retrieved for this statement, each representing a data point.
* **Methods:**
  + \_\_init\_\_(self, statement\_type): Initializes the statement type.
  + add\_concept(self, concept): Adds a discovered concept to the concepts list.
  + add\_fact(self, fact): Adds a retrieved fact to the facts list.
  + get\_fact(self, concept\_name): A helper method to easily retrieve a specific fact (e.g., "Revenues").
  + to\_dataframe(self): A useful method to convert the facts into a Pandas DataFrame for easy analysis.

This class structure mirrors the hierarchy of financial data: a Company has Reports, which contain FinancialStatements, which are composed of Concepts and Facts. This design makes your code intuitive, modular, and easy to maintain and expand.