**Overview of the Custom Class 'CellDetails'**

In this project, wherever string manipulation is performed, we need to replace it with the name of our custom class. For example, in the code, you will find various string manipulations like **cell\_name[1:8]** for site\_name, **cell\_name[1:9]** for site\_name, **cell\_name[1:-2]** for site\_name, **cell\_name[1:10]** for sector\_name, etc.

Our task is to identify these instances and replace them with data from our custom function. This approach will have several benefits, such as eliminating repetitive string manipulation and enabling our function to retrieve data from the cache, improving efficiency.

Currently, the class I've created is not fully accurate. It may require modifications based on new challenges or ideas that arise.

**Class Name: 'CellDetails'**

The 'CellDetails' class provides essential information, such as **site\_name, sector\_name**, and **band\_name**, from site data (target\_data) and cell data. Additionally, it caches the data to avoid repeated retrieval. If the required data is not found, the class will retrieve the data from enetsdk and return it.

### **Overview**

The CellDetails class is designed to manage and cache key details related to cellular nodes, such as the node name, site name, sector, and band. This class utilizes the enetsdk library to interact with a database of cell details and provides methods to retrieve and store information related to cells in a cache, improving efficiency and reducing redundant calculations.

### **Class Breakdown**

1. **\_\_init\_\_(self, script\_data)**:
   * This is the initializer method where the class is instantiated.
   * **Parameters**:
     + script\_data: An object containing the necessary methods for retrieving data. It's used to interact with other systems (e.g., enetsdk and get\_cm\_reader).
   * **Attributes**:
     + cache: A dictionary to store cached cell details, avoiding repeated data retrieval.
     + script\_data: The script data object passed to interact with external data sources.
     + cdb: The enetsdk's GetCellDB() method, which helps retrieve cell-related data.
2. **run\_cache(self, targets: list[object] = None, cells: list[str] = None)**:
   * This method caches cell details for a list of target objects or cell names.
   * **Parameters**:
     + targets: An optional list of target objects representing cells.
     + cells: An optional list of cell names (strings) to resolve and process.
   * If cells are provided, it retrieves the corresponding target objects and extends the list of targets, which is then passed to the get\_cell\_details() method to build a cache of cell data.
3. **\_validate\_and\_get\_cell\_name(self, target: object, cell: str) -> str**:
   * This helper function ensures that either the target or cell is provided, but not both or neither. It returns the cell name.
   * **Parameters**:
     + target: The target object representing a cell.
     + cell: The cell name (string).
   * **Returns**: The cell name as a string.
4. **\_get\_cached\_value(self, cell\_name: str, key: str)**:
   * This method retrieves a specific value from the cache based on the cell\_name and key (like node name, site name, etc.).
   * If the data is not found in the cache, it fetches the required details from enetsdk and updates the cache.
   * **Parameters**:
     + cell\_name: The name of the cell.
     + key: The specific key (e.g., nodeName, siteName, etc.) to retrieve.
   * **Returns**: The value associated with the key, or None if not found.
5. **get\_node\_name(self, target: object = None, cell: str = None) -> str**:
   * Retrieves the node name for a given cell.
   * **Parameters**:
     + target: The target object representing the cell.
     + cell: The cell name.
   * **Returns**: The node name for the cell, or None if not found.
6. **get\_site\_name(self, target: object = None, cell: str = None) -> str**:
   * Retrieves the site name for a given cell.
   * **Parameters**:
     + target: The target object representing the cell.
     + cell: The cell name.
   * **Returns**: The site name for the cell, or None if not found.
7. **get\_sector(self, target: object = None, cell: str = None) -> str**:
   * Retrieves the sector for a given cell.
   * **Parameters**:
     + target: The target object representing the cell.
     + cell: The cell name.
   * **Returns**: The sector for the cell, or None if not found.
8. **get\_band(self, target: object = None, cell: str = None) -> str**:
   * Retrieves the band for a given cell.
   * **Parameters**:
     + target: The target object representing the cell.
     + cell: The cell name.
   * **Returns**: The band for the cell, or None if not found.
9. **get\_cell\_details(self, targets: list[object]) -> dict**:
   * This method processes a list of target objects and returns a dictionary containing detailed information about the cells, such as nodeName, siteName, sector, and band.
   * **Parameters**:
     + targets: A list of target objects (cells) to process.
   * **Returns**: A dictionary with cell names as keys and cell details (node name, site name, sector, band) as values.
   * **How It Works**:
     + First, it retrieves all site names for the given targets using get\_cm\_reader().read\_cells\_by\_uids().
     + Then, it processes each target:
       - Retrieves the cell name.
       - Determines the node name based on the technology (NR, LTE, 4G-NBIoT).
       - Retrieves the site name using the UID.
       - Derives the sector and band.
     + Finally, it stores all the retrieved data in a dictionary and returns it.

### **Key Features and Benefits**

* **Caching**: The class stores the data in a cache to avoid redundant operations.
* **Efficiency**: Data retrieval is optimized by directly accessing cached values, reducing the need for repetitive calculations.
* **Flexibility**: It can handle both target objects and cell names, automatically resolving and processing them.
* **Error Handling**: Exceptions are caught when retrieving site names or determining the band, ensuring the class handles errors gracefully.

### **Conclusion**

The CellDetails class simplifies and optimizes the process of extracting and caching cell details like node name, site name, sector, and band, while avoiding the need for repetitive string manipulations. By using caching and error handling, it enhances performance and reliability in retrieving and managing cell data.