1) Class Description:

The TagManager class is designed to manage string-based tags associated with various entities or entries. Its primary goal is to allow users to efficiently track, add, remove, and retrieve entries based on tags. The class should be capable of associating one or more tags with each entry, allowing for quick lookup of all entries associated with a specific tag. The high-level functionality should include adding tags to entries, removing tags, retrieving all tags associated with an entry, retrieving all entries associated with a tag, and ensuring fast lookups and efficient memory usage.

Goals:

- Enable quick access to all entries associated with a given tag.
- Allow entries to have one or more associated tags.
- Support efficient tag addition and removal.
- Handle edge cases like removing non-existent tags or querying tags that don't exist.

High-level Functionality:

- **Add a tag**: Attach a tag to an entry.
- **Remove a tag**: Detach a tag from an entry.
- **Retrieve tags for an entry**: List all tags associated with a given entry.
- **Retrieve entries for a tag**: Get all entries associated with a specific tag.
- **Clear all tags for an entry**: Remove all tags associated with an entry.
- **Clear all entries for a tag**: Remove all associations with a specific tag.

2) Similar Classes in the Standard Library:

- **set**: Since tags will need to be unique for each entry, a set will be useful for ensuring there are no duplicates.
- **dict**: A dictionary will help map each tag to a list of entries, enabling fast access and modification.
- **collections.Counter**: Can inform how we think about counting occurrences or managing frequency

3) Key Functions to Implement:

1. **add_tag(entry, tag)**: Adds a tag to the entry.
    - Parameters: entry (str), tag (str)
2. **remove_tag(entry, tag)**: Removes a tag from an entry.
    - Parameters: entry (str), tag (str)
3. **get_tags(entry)**: Retrieves all tags associated with an entry.
    - Parameters: entry (str)
    - Returns: A list or set of tags.
4. **get_entries(tag)**: Retrieves all entries associated with a specific tag.
    - Parameters: tag (str)
    - Returns: A list or set of entries.
5. **clear_entry_tags(entry)**: Clears all tags for a specific entry.
    - Parameters: entry (str)
6. **clear_tag_entries(tag)**: Clears all entries associated with a specific tag.
    - Parameters: tag (str)
    - Effect: Removes all entries from the given tag.
7. **has_tag(entry, tag)**: Checks if an entry has a specific tag.
    - Parameters: entry (str), tag (str)

- Returns: Boolean indicating whether the entry is associated with the tag.

4) Error Conditions:

- **Tag or entry not found**:
  - **Condition**: Attempting to remove a tag from an entry that doesn't have it or querying entries for a non-existent tag.
  - **Response**:
    - **Type**: User error
    - **Handling**: Return a special condition (like None or an empty list) to indicate the tag/entry does not exist. This can be handled with a try-except block or simple checks.
- **Invalid input types**:
  - **Condition**: Non-string input for entry or tag.
  - **Response**:
    - **Type**: Programmer error
    - **Handling**: Use an assert statement to verify input types and raise an AssertionError if invalid.
- **Memory constraints**:
  - **Condition**: The collection of tags or entries exceeds available memory (don't think this would happen but possibly!)
  - **Response**:
    - **Type**: Recoverable error
    - **Handling**: Raise a MemoryError or other suitable exception if memory limitations are encountered.

5) Expected Challenges:

- Efficient data storage
- Handling edge cases:
  - what happens when an entry has no tags, or a tag has no entries?
  - Properly returning empty lists or sets is important,
  - considering whether to raise exceptions in such cases is something that needs to be thought through.
- Testing

6) Other Class Projects from Other Groups:

- No seeing a similar one ATM