**Limitation of naive solution:**

* Massive data duplication: the user information is duplicated whenever they add a new comment, which results in redundancy in memory usage.
* Inefficient updates: When a user updates something about their profile, we must loop through all of their comments to update the corresponding information

**Flyweight pattern:**

The goal is to avoid duplicating identical data by letting everyone share a single object instead of copying it.  
We split our original Comment into two disjoint classes:  
1. **CommentContent** – holds the text (unique to each comment).  
2. **CommenterInfo** – holds the profile data (name, avatar URL, profile URL) shared by all comments from the same user.

Each user maps to exactly one CommenterInfo instance, so you never store the same profile data more than once.

**Note:**

* For simplicity, the key obtained by concatenating all 3 strings is used, however, one can use other valid hash functions to make the process of finding the key faster, using std::hash is also a good option.

**Pros and Cons:**

* **Pros:**
* huge memory savings when many objects share identical data.
* Faster updates, since only 1 instance is used for each user’s information.
* **Cons:**
* Increase code complexity
* Increase computational cost (For example, each time we create a comment, instead of simply initializing it like in the naive code, we need to hash to find a key, find it in map and return corresponding SharedState).

**When to use/ not to use:**

* Consider using flyweight when a small set of shared data (e.g. user profiles) backs a huge number of objects (e.g. comments)
* Not to use flyweight when nearly every object has unique shared fields
* Not to use when the shared states change frequently, making the factory map remove and re-insert frequently.