**3. Document Design Decisions**

**Ideea principală**  
Am vrut să fac un program clar și ușor de citit, care:

* citește cuvintele dintr-un fișier;
* le grupează pe anagrame;
* păstrează ordinea în care apare primul cuvânt din fiecare grup.

**Alegerea structurii de date**  
Pentru asta am folosit **LinkedHashMap**, care reține ordinea de inserare.

**Cum formez cheia fiecărui grup**

* Transform cuvântul în litere mici;
* Îl descompun în caractere;
* Sortez caracterele.  
  Astfel, *"Roma"* și *"amor"* vor ajunge în același grup.

**Citirea fișierului**

* Fișierul e citit linie cu linie folosind **BufferedReader** și **UTF-8** pentru compatibilitate și performanță.
* Procesarea este *streaming*, adică nu țin tot fișierul în memorie deodată.

**Sortarea finală**  
La final, sortez cuvintele din fiecare grup fără să țin cont de majuscule, pentru afișare mai curată.

**Dependențe**

* Nu am folosit biblioteci externe — totul e din **JDK**.
* Funcția care face cheia este separată, astfel încât poate fi modificată ușor fără să afecteze restul programului.

**4. Scalability Considerations**

**Pentru ~10 milioane de cuvinte**

Soluția actuală ar funcționa cu mici optimizări:

* Buffer de citire mai mare;
* Procesare pe bucăți (*chunking*);
* Salvare temporară pe disc dacă RAM-ul nu ajunge;
* Înlocuirea sortării caracterelor cu o metodă bazată pe frecvența literelor (mai rapidă și mai economică în memorie);
* Procesare în paralel pe mai multe *thread*-uri, combinând rezultatele la final și păstrând ordinea grupurilor.

**Pentru ~100 miliarde de cuvinte**

Ar fi nevoie de procesare distribuită (ex. **Apache Spark** sau **MapReduce**):

* Împărțirea fișierului între mai multe mașini;
* Gruparea locală pe fiecare nod;
* Păstrarea aceleiași chei (bazată pe frecvența literelor);
* Ordinea grupurilor determinată de poziția primei apariții globale;
* Stocare și îmbinare finală în **HDFS** sau cloud;
* Procesare în paralel pentru volum mare și toleranță la erori.