TRANSLITERATION GUIDE BASED ON ISO15919

OVERVIEW:

This Python program is designed to transliterate Hindi text into a phonetic English equivalent and compare the phonetic similarity of word pairs. It reads the input from a database, processes it, and outputs results similar to those of the backend.

KEY COMPONENTS:

1. Input Handling

- ✓ The program reads pairs of Hindi strings (e.g., ह and क) from a file.
- ✓ Each line is processed to extract two target strings (h and k).
- ✓ Example:
- ✓ Input line: "ह\tक"
- ✓ Extracted: h = "ह", k = "क"

2. Phonetic Transliteration

The goal is to convert Hindi words into their English phonetic equivalents by mapping Hindi characters to Latin (English) characters.

Key Features in Transliteration:

- ✓ Consonants and Vowels Mapping:
- ✓ Each Hindi character (e.g., क, ख, ग) is mapped to its phonetic English counterpart (e.g., ka, kha, ga).
- ✓ Example: हिमालय → himala
- ✓ Handling Matras (e.g., ा, ि, ी):
- ✓ Matras modify vowels and consonants, changing their pronunciation.
- ✓ Example:
- ✓ क (ka) + ा = का (kaa)
- ✓ क (ka) + ि = कि (ki)
- ✓ Halant (्):
- ✓ The halant removes inherent vowels from consonants and connects two consonants to form clusters.
- ✓ Example:
- ✓ क् + ष = क्ष (ksha)
- ✓ It also prevents duplicate sounds by eliminating default vowel sounds.
- ✓ Clusters and Conditional Rules:

- ✓ Consonant clusters (like ज्ञ or त्र) are handled by specific rules.
- ✓ Example:
- ✓ ज्ञ = gya or jnya

3. Comparison Logic

- ✓ Once the two Hindi strings are transliterated into English phonetic forms, they are compared.
- ✓ If it is equal, the words are considered phonetically similar.
- ✓ Otherwise, they are different.

4. Output

- ✓ Results are stored in the database and are used for fuzzy matching.

ISO COMPLIANCE

ISO 15919 is a widely recognized standard for transliterating Indic scripts to Latin characters.

Current Compliance in Your Code:

- Basic mappings from Hindi characters to Latin equivalents are present.
- Vowel and consonant distinctions are recognized, and matras are partially handled.

To Achieve Better Compliance:

1.Standardized Mappings:

- ❖ Use precise mappings defined by ISO 15919.
- ❖ Example:
- ❖ Instead of sha, map श to ś (ISO standard representation).
- ❖ Example Table:

श -> ś

ष -> ṣ

स -> s

2. Unicode Normalization:

- ❖ Handle combining characters (like ◌ँ, ◌ं) properly to maintain phonetic accuracy.
- ❖ Normalize Unicode inputs to avoid mismatched representations.

3. Error Handling:

- Gracefully handle invalid inputs or missing characters with user-friendly messages.
- Example:
- Input: "123"
- Output: "Error: Non-Hindi characters found."

CODE OPTIMIZATION SUGGESTIONS

1. Refactor Repetitive Logic:

- Instead of a repetitive switch or if statements, use a map-based approach for character-to-sound mapping.
- Example:

```
Map<Character, String> translitMap = Map.of('क', "ka", 'ख', "kha", 'ग', "ga", ...);
```

2. Matra Handling:

- Use a dedicated data structure (e.g., a map or list) to dynamically handle matras.
- Example:

```
Map<Character, String> matraMap = Map.of( 'ा', "aa", 'ि', "i", 'ी', "ee", …);
```

3. Reusable Transliteration Function:

- Consolidate logic for transliterating h and k into a single reusable function (transliterate).
- Example:

```
public static String transliterate(String input) {
 // Character mappings (consonants, vowels, matras)
 Map<Character, String> translitMap = Map.of(...);
   StringBuilder result = new StringBuilder();
 for (char c : input.toCharArray()) {
            result.append(translitMap.getOrDefault(c, ""));
  }
 return result.toString();
}
```

4. Advanced Error Handling:

- Handle edge cases:

- Empty strings ("") → Output: "Error: Input is empty."
- Non-Hindi inputs ("abc") → Output: "Error: Invalid characters found."

PHONETIC SIMILARITY ENHANCEMENTS

1. Approximate Matching:

- Implement algorithms like Levenshtein Distance or Soundex for better similarity detection.
- These algorithms allow a tolerance for minor variations in spelling or transliteration.

2. Weighting System:

- Assign different weights to vowels, consonants, and matras based on their phonetic significance.
- Example:
- Exact match of consonants: Weight = 2
- Similar vowels: Weight = 1

PROPOSED TRANSLITERATION FUNCTION

A refactored version of your transliteration function:

```
public static String transliterate(String input) {

  Map<Character, String> translitMap = Map.of(

    'क', "ka", 'ख', "kha", 'ग', "ga", 'घ', "gha",

    'च', "cha", 'छ', "chha", 'ज', "ja", 'झ', "jha",

    'ट', "ta", 'ठ', "tha", 'ड', "da", 'ढ', "dha",

    'ण', "na", 'त', "tta", 'थ', "ttha", 'द', "dda",

    'ध', "ddha", 'न', "na", 'प', "pa", 'ब', "ba",

    'फ', "pha", 'भ', "bha", 'म', "ma", 'य', "ya",

    'र', "ra", 'ल', "la", 'व', "va", 'ह', "ha",

    'स', "sa", 'श', "sha", 'ष', "sha", 'अ', "a",

    'आ', "aa", 'इ', "i", 'ई', "ee", 'उ', "u",

    'ऊ', "uu", 'ए', "e", 'ऐ', "ai", 'ओ', "o",

    'औ', "au" );

 StringBuilder result = new StringBuilder();

  for (char c : input.toCharArray()) {
    result.append(translitMap.getOrDefault(c, ""));}
return result.toString();
}
```

ADVANCED ENHANCEMENTS

1. Integration with NLP Libraries:

- Use libraries like ICU4J for transliteration and normalization.
- Example: The Transliterator class in ICU4J handles Indic scripts with ease.

2. Extended Phonetic Comparison:

- Implement Soundex, Metaphone, or other algorithms to enhance phonetic similarity detection.