Project 2

**Project Name: Link Shortener Development Date:24-06-24**

**Name: Hasanapuram Anil kumar**

Requirements and Features:

* Use Java as the primary programming language.
* Create a class or set of classes that manage the shortening and expanding of URLs.
* Implement a basic hash function for generating short URLs.
* Include error handling to address scenarios like duplicate long URLs and invalid short URLs.
* optionally, consider persisting data to maintain link mappings between sessions.
* Develop a simple command-line interface (CLI) or web-based interface for user interaction.

**package** oops;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Scanner;

**public** **class** Project

{

**private** Map<String, String> shortToLongMap;

**private** Map<String, String> longToShortMap;

**public** Project()

{

**this**.shortToLongMap = **new** HashMap<>();

**this**.longToShortMap = **new** HashMap<>();

}

// Method to generate a short URL using SHA-256 hash function

**private** String generateShortURL(String longURL)

{

**Try**

{

MessageDigest md = MessageDigest.*getInstance*("SHA-256");

**byte**[] hashBytes = md.digest(longURL.getBytes());

StringBuilder sb = **new** StringBuilder();

**for** (**byte** b : hashBytes)

{

sb.append(String.*format*("%02x", b));

}

// Take the first 8 characters as the short URL

**return** sb.substring(0, 8);

}

**catch** (NoSuchAlgorithmException e)

{

e.printStackTrace();

**return** **null**;

}

}

// Method to shorten a long URL

**public** String shortenURL(String longURL)

{

**if** (longToShortMap.containsKey(longURL))

{

**return** longToShortMap.get(longURL);

}

String shortURL = generateShortURL(longURL);

shortToLongMap.put(shortURL, longURL);

longToShortMap.put(longURL, shortURL);

**return** shortURL;

}

// Method to expand a short URL to the original long URL

**public** String expandURL(String shortURL)

{

**if** (shortToLongMap.containsKey(shortURL))

{

**return** shortToLongMap.get(shortURL);

} **else** {

**return** "Invalid short URL!";

}

}

**public** **static** **void** main(String[] args)

{

Project shortener = **new** Project();

Scanner scanner = **new** Scanner(System.***in***);

**while** (**true**)

{

System.***out***.println("Enter a command ('shorten' or 'expand', 'exit' to quit):");

String command = scanner.nextLine().trim().toLowerCase();

**if** ("exit".equals(command))

{

System.***out***.println("Exiting Link Shortener...");

**break**;

} **else** **if** ("shorten".equals(command))

{

System.***out***.println("Enter the long URL to shorten:");

String longURL = scanner.nextLine().trim();

String shortURL = shortener.shortenURL(longURL);

System.***out***.println("Shortened URL: " + shortURL);

} **else** **if** ("expand".equals(command))

{

System.***out***.println("Enter the short URL to expand:");

String shortURL = scanner.nextLine().trim();

String longURL = shortener.expandURL(shortURL);

System.***out***.println("Expanded URL: " + longURL);

}

**else**

{

System.***out***.println("Invalid command. Please enter 'shorten', 'expand', or 'exit'.");

}

System.***out***.println(); // Empty line for readability

}

scanner.close();

}

}

**Output:**

Enter a command ('shorten' or 'expand', 'exit' to quit):

exict

Invalid command. Please enter 'shorten', 'expand', or 'exit'.

Enter a command ('shorten' or 'expand', 'exit' to quit):

shorten

Enter the long URL to shorten:

Shortened URL for https://www.example.com/article/12345 is: eb8065a1

Shortened URL: d3653df0

Enter a command ('shorten' or 'expand', 'exit' to quit):

expand

Enter the short URL to expand:

Shortened URL for https://www.example.com/article/12345 is: eb8065a1

Expanded URL: Invalid short URL!

Enter a command ('shorten' or 'expand', 'exit' to quit):

**Explanation:**

1. **LinkShortener Class**:
   * **Instance Variables**: shortToLongMap and longToShortMap are used to store mappings between short URLs and long URLs, ensuring bidirectional lookup.
2. **generateShortURL Method**:
   * Uses SHA-256 hashing to generate a short URL from the given long URL. This method ensures that each long URL will produce a unique short URL.
3. **shortenURL Method**:
   * Checks if the long URL has already been shortened (exists in longToShortMap). If yes, returns the existing short URL; otherwise, generates a new short URL using generateShortURL.
4. **expandURL Method**:
   * Checks if the provided short URL exists in shortToLongMap. If yes, returns the corresponding long URL; otherwise, returns an error message indicating an invalid short URL.
5. **Main Method**:
   * Implements a simple command-line interface (CLI) using Scanner for user interaction.
   * Accepts commands ('shorten' or 'expand') from the user.
   * 'shorten' command prompts the user to enter a long URL and prints the generated short URL.
   * 'expand' command prompts the user to enter a short URL and prints the corresponding long URL.
   * 'exit' command terminates the application loop.

**Usage:**

* Compile the above Java code and run the LinkShortener class.
* Follow the prompts in the console to shorten or expand URLs.

This implementation provides a foundational example of a Link Shortener application in Java, demonstrating key aspects such as hashing, map-based storage, error handling, and basic command-line interface (CLI) interaction.