JAVA COMMAND-LINE UTILITY FOR TEXT MANIPULATION, FILE ENCRYPTION AND FILE COMPRESSION

PROJECT OVERVIEW

The Java Command-Line Utility is a comprehensive tool designed to perform various essential tasks, including text manipulation, file encryption, and file compression along with corresponding test cases using JUnit. Built using the robust and versatile Java programming language, this utility is intended to run seamlessly in a terminal or command prompt environment. It aims to provide a straightforward yet powerful interface for users to accomplish these tasks efficiently and effectively.

Key Features

The utility encompasses three main functionalities:

1. **Text Manipulation:**

   This feature allows users to perform basic text operations, such as reversing strings. Text manipulation is crucial in many applications, from simple data formatting to complex data processing tasks. By providing a command to reverse text, the utility helps users quickly modify text data as needed.

2. **File Encryption:**

   Security is a significant concern in today's digital world. The file encryption feature ensures that sensitive information can be securely stored and transmitted. Using the Advanced Encryption Standard (AES) algorithm, users can encrypt files with a specified key, protecting their data from unauthorized access. This feature is particularly useful for safeguarding confidential documents and sensitive information.

3. **File Compression:**

   Efficient storage and transfer of files often require compression to reduce file size. The utility includes a file compression feature using the GZIP format, which helps users compress large files, making them easier store and share. This functionality is essential for managing disk space and optimizing file transfer speeds.

PROJECT TEAM

1. Harshitha Botta (Team leader)
2. Sachin Nagnath Isamantri
3. Rajalakshmy N.V
4. Ashraf Khan
5. Shivakrishna Erumalla

PROJECT TECHNOLOGIES

- **Programming Language**
  - Java: The primary language for developing the command-line utility.

- **Development Environment**
  - Java Development Kit (JDK): Install Version 8 or higher required to compile and run the Java code.
  - Integrated Development Environment (IDE): Tools like IntelliJ IDEA, Eclipse, etc.

- **Command-Line Interface (CLI)**
  - Terminal or Command Prompt: Environment where the utility runs and interacts with users.

- **Testing**
  - JUnit: Utilized for writing and running unit tests.
  - **JUnit**: Version 4.12
  - **Hamcrest**: Version 1.3

PROJECT SETUP

Create Project Directory: Create a directory for your project.

For example:

- o " C:\Users\<YourUsername>\Desktop\CommandLineTool "

Download Dependencies: Download junit-4.12.jar and hamcrest-core-1.3.jar and place them in your project directory.

- o JUnit 4.12
- o Hamcrest 1.3

Command Syntax

Compiling the Code :

- **Navigate to the Project Directory:**
  - o cd C:\Users\<YourUsername>\Desktop\CommandLineTool
- Use the following command to compile all Java files:
  - o " javac *.java "
- Command to compile java test files:
  - o " javac -cp .;"C:\Users\<YourUsername>\Desktop\CommandLineTool\junit-4.12.jar";"C:\Users\<YourUsername>\Desktop\CommandLineTool\hamcrest-core-1.3.jar" *.java "

Running the Code :

- The command java CommandLineTool is used to run the main class CommandLineTool of your Java project. If the CommandLineTool class contains a main method, this command will execute that method. The exact behavior depends on how the 'main'method is implemented in CommandLineTool.

- " java CommandLineTool"

Commands to run the specific functions or tasks :

- To find and replace :
  - java CommandLineTool findreplace -find world -replace Java path/to/yourfile.txt

- To Convert into upper case:
  - java CommandLineTool toupper path/to/yourfile.txt

- To Concert into lower case:
  - java CommandLineTool tolower path/to/yourfile.txt

- To count the words,lines :
  - java CommandLineTool count path/to/yourfile.txt

- To compress the file :
  - java CommandLineTool compress path/to/yourfile.txt

- To decompress the file :
  - java CommandLineTool decompress path/to/yourfile.zip

- To Encrypt the file:
  - java CommandLineTool encrypt password path/to/yourfile.txt

- To Decrypt the file:
  - java CommandLineTool decrypt password path/to/yourfile.txt

Commands for Testing with JUnit :

- After compilation, you can run your JUnit tests using the java command:

   - “ java -cp .;junit-4.12.jar;hamcrest-core-1.3.jar org.junit.runner.JUnitCore TextManipulatorTest FileCompressorTest FileEncryptorTest ”

PROJECT REFERENCES

- https://www.theserverside.com/video/Java-command-line-tools-every-JDK-developer-should-know
- https://www.javatpoint.com/junit-tutorial
- https://youtu.be/RqTd-BtC2sY?si=3RHrLIWe3N3tBAvL
- https://youtu.be/8wlE6DgOWBs?si=EziWjlBupvMUJBlu
- https://youtu.be/5Dkw0Yl82JQ?si=nxJ5v3uk2pom1Ddv

PROJECT RISKS

1. Requirement Misalignment

Risk: Misunderstanding or incomplete requirements can lead to a utility that does not meet user needs. **Mitigation:**

- Conduct thorough requirement analysis and validation sessions with stakeholders.
- Regularly review and validate requirements throughout the project lifecycle.
- Create clear and comprehensive documentation.

2. Technical Complexity

Risk: The complexity of implementing encryption algorithms, file compression, and text manipulation may lead to unforeseen technical challenges. **Mitigation:**

- Use established libraries and frameworks for complex tasks (e.g., Apache Commons, Bouncy Castle).
- Conduct proof-of-concept (PoC) implementations for complex features.
- Provide adequate time for research and development.

3. Performance Issues

Risk: The utility might perform poorly with large files or complex operations, leading to slow execution and poor user experience. **Mitigation:**

- Optimize code for performance and efficiency.
- Conduct performance testing and profiling.
- Implement performance benchmarks and optimize based on findings.

4. Security Vulnerabilities

Risk: Security vulnerabilities in encryption/decryption functionalities could lead to data breaches or loss. **Mitigation:**

- Follow security best practices for handling sensitive data.
- Regularly review and update encryption libraries.
- Conduct security testing and code reviews focusing on vulnerabilities.

5. Error Handling

Risk: Poor error handling may lead to unclear error messages or application crashes, confusing users. **Mitigation:**

- Implement comprehensive error handling and logging.
- Provide informative and user-friendly error messages.
- Conduct extensive testing to cover edge cases and unexpected inputs.