**Java Project 2: Basic File Encryption/Decryption**

# Initial prompt

I am an undergraduate student that studies information technology. I do not know much about programming and would like to make a Java project to help build my knowledge. Can you be an assistant to help me make a project. I will provide more information in later prompts. Also clarify throughout the process if what was outputted okay. Just make a small short response.

# Prompt 2

I am nearly going to graduate and need to do 1 more advanced IT subject. I was advised to take a subject 1 semester as I couldn’t take a leave of absence.

“If you pass your course in semester 2, 2024, you will be left with 1 more advanced IT option course. You can choose either of

Cloud Security or Blockchain fundamentals for this last course. Here are the course guides:

• Cloud Security: http://www1.rmit.edu.au/courses/038407

• Blockchain fundamentals: http://www1.rmit.edu.au/courses/INTE26272450

You mentioned that you want to take LoA for semester 1, because you want to take System Deployment & Operations (COSC2759).”

- Shekhar Kalra

Here is the course information for Cloud Security, one of the courses I need to do enclosed in brackets:

{

Course Title: Cloud Security

Credit Points: 12.00

Course Description

Cloud security is an evolving sub-domain of computer security, network security, and, more broadly, information security. It refers to a broad set of technologies deployed to protect data, applications, and the associated infrastructure of cloud computing.

This course provides a practical survey of both the principles and practice of cloud security. The emphasis of this course is on the underlying principles and techniques of cloud security with examples of how they are applied in practice.

Please note that if you take this course for a bachelor honours program, your overall mark in this course will be one of the course marks that will be used to calculate the weighted average mark (WAM) that will determine your award level. This applies to students who commence enrolment in a bachelor honours program from 1 January 2016 onwards.

Program Learning Outcomes

This course is an option course and not a core course, so that it does not need to contribute to the Program Learning Outcomes.

Course Learning Outcomes

Upon successful completion of this course you should be able to:

1. Demonstrate knowledge of cloud security principles and mechanisms

2. Demonstrate computer programming and configuration skills required to develop a cloud security infrastructure

3. Identify cloud security weaknesses by recognising and discovering threats and vulnerabilities to cloud computing

4. Problem solve how to fix cloud security weaknesses and mitigate security threats to cloud computing

5. Demonstrate knowledge and skills to prepare for industry cloud security certificate exams e.g. CCSK, CCSP

6. Communicate clearly and effectively using the technical language of the field correctly

Overview of Learning Activities

The learning activities included in this course are:

• Lectorials: Key concepts will be explained in lectorials in which course material will be presented and the subject matter will be illustrated with demonstrations and examples;

• Tutorial sessions focus on analysing and problem solving of given scenarios, such as security analysis and secure solutions;

• Computer laboratory sessions provide practices in the application of security techniques.

}

Here is information from one of tutors about what I should to prepare for the subject enclosed in brackets.

{

The programming tasks for this course are expected to be completed in Java or JavaScript. Generally speaking, and without going into too much detail, we use a programming language to apply a cryptographic concept, where the objective is to transform data into a form that appears random, but the original data can be recovered.

As a simplified example of this idea, consider that we have a character string like "Hello World!" and the task is to transform each character, such that the transformation can be reversed. For this task, the programming constructs we would need to use are:

* **String handling**: obtaining individual characters.
* **Basic data types**: understanding the relationship between character and integer data types.
* **IF statement:** can be used to handle characters like '!' and the space character as special cases.
* **FOR loop:** need a way to iterate over all characters in a string.
* **Arithmetic:** need to use a mathematical operation that can be reversed.

These constructs are found in either Java or JavaScript. In the case of JavaScript, we also need an API that allows us to get input from a webpage to make the implementation more interactive (that is, the Document Object Model that represents the HTML document). We typically avoid the use of language features like polymorphism or exception handling, because this can distract from the task at hand. For more advanced algorithms, we make use of programming libraries, which are provided as needed.

These constructs are found in either Java or JavaScript. In the case of JavaScript, we also need an API that allows us to get input from a webpage to make the implementation more interactive (that is, the Document Object Model that represents the HTML document). We typically avoid the use of language features like polymorphism or exception handling, because this can distract from the task at hand. For more advanced algorithms, we make use of programming libraries, which are provided as needed.

} Does this make sense. Just answer yes or no.

# Prompt 4

I was thinking of doing a project like this:

Basic File Encryption/Decryption

Difficulty: Easy

Goal: Create a program that reads a text file, applies a simple encryption method (e.g., Caesar cipher), and writes the encrypted output to another file. Include a decryption option.

Skills Covered:

*  File handling
*  String manipulation
*  Loops and conditionals Does that seem appropriate?

# Prompt 5

In that case can you provide me a deadline of 2 weeks, the due date and details of project.

**Details of project**

Sure! Here's the information for your project:

**Project Details: Basic File Encryption/Decryption**

**Goal:**  
Create a program in Java that:

1. Reads a text file.
2. Applies a simple encryption method (e.g., Caesar cipher).
3. Writes the encrypted output to another file.
4. Provides a decryption option to reverse the process.

**Key Features:**

* Input: User specifies the file to encrypt/decrypt and the shift key for the cipher.
* Output: Encrypted/Decrypted text is written to a new file.
* Options: User can choose between encryption or decryption.

**Skills Covered:**

* File handling.
* String manipulation.
* Loops and conditionals.
* Basic command-line interaction (optional).

**Deadline:**  
**Two weeks from today: December 15, 2024.**

**Tasks to Complete:**

1. **Day 1-3:** Learn or review file handling in Java.
   * Practice reading and writing text files.
   * Ensure your development environment is set up.
2. **Day 4-7:** Implement basic Caesar cipher logic.
   * Create functions for encryption and decryption.
3. **Day 8-10:** Integrate file handling and encryption logic.
   * Allow the program to read from and write to files.
   * Test using small text files.
4. **Day 11-13:** Add a user-friendly menu for interaction.
   * Let users choose encryption or decryption, input shift key, etc.
5. **Day 14:** Final testing and polish.

# Prompt 6

I am using w3schools to look at Java concepts. Here is all the concepts they cover. Note that dot points are a subheading of the heading:

**Java Basics**

Java Syntax

Java Output

* Print Text
* Print Numbers

Java Comments

* Java Variables
* Print Variables
* Multiple Variables

Java Data Types

* Data Types Numbers
* Booleans
* Characters
* Real-Life Example
* Non-primitive Types

Java Type Casting

Java Operators

Java Strings

* Strings
* Concatenation
* Numbers and strings
* Special characters

Java Math

Java Booleans

Java If...Else

* If
* Else
* Else if
* Short hand.. if else (**ternary operator)**

Java Switch

Java While Loop

* Do/ while loop

Java For Loop

* For loop
* Nested loops
* For-each loop

Java Break/Continue

Java Arrays

* Loop through an array
* Multidimensional arrays

**Java Methods**

What are Java Methods

Java Method Parameters

* Parameters
* Return values

Java Method Overloading

Java Scope

Java Recursion

**Java Classes**

Java OOP

Java Classes/Objects

Java Class Attributes

Java Class Methods

Java Constructors

Java Modifiers

Java Encapsulation

Java Packages / API

Java Inheritance

Java Polymorphism

Java Inner Classes

Java Abstraction

Java Interface

Java Enums

Java User Input

Java Date

Java ArrayList

Java LinkedList

Java List Sorting

Java HashMap

Java HashSet

Java Iterator

Java Wrapper Classes

Java Exceptions

Java RegEx Java Threads

Java Lambda

Java Advanced Sorting

**Java File Handling**

Java Files

Java Create/Write Files

Java Read Files

Java Delete Files

**Java How To's**

Add Two Numbers

Count Words

Reverse a String

Sum of Array Elements

Convert String to Array

Sort an Array

Find Array Average

Find Smallest Element

ArrayList Loop

HashMap Loop

Loop Through an Enum

Area of Rectangle

Even or Odd Number

Positive or Negative

Square Root

Random Number

**Java Reference**

Java Reference

Java Keywords

* assert
* abstract
* boolean
* break
* byte
* case
* catch
* char
* class
* continue
* default
* do
* double
* else
* enum
* exports
* extends
* final
* finally
* float
* for
* if
* implements
* import
* instanceof
* int
* interface
* long
* module
* native
* new
* package
* private
* protected
* public
* return
* requires
* short
* static
* super
* switch
* synchronized
* this
* throw
* throws
* transient
* try
* var
* void
* volatile
* while

Java String Methods

* charAt()
* codePointAt()
* codePointBefore()
* codePointCount()
* compareTo()
* compareTolgnoreCase()
* concat() contains()
* contentEquals()
* copyValueOf(
* endsWith()
* equals()
* equalsIgnoreCase()
* format() getBytes()
* getChars()
* hashCode()
* indexOf()
* isEmpty(
* join()
* lastIndexOf()
* length()
* matches()
* offsetByCodePoints()
* regionMatches()
* replace()
* replaceAll()
* replaceFirst()
* split()
* startsWith(
* subSequence()
* substring()
* toCharArray()
* toLowerCase()
* toString()
* toUpperCase()
* trim()
* valueOf()

Java Math Methods

* abs()
* acos()
* addExact()
* asin()
* atan()
* atan20
* cbrt()
* ceil()
* copySign()
* cos()
* cosh()
* decrementExact()
* exp0
* expm10
* floor()
* floorDiv0
* floorMod()
* getExponent()
* hypot()
* IEEEremainder()
* incrementExact()
* logo
* log100)
* log1p0
* max0
* mino
* multiplyExact()
* negateExact()
* nextAafter()
* nextDown()
* nextUp()
* pow()
* random()
* rint(
* round()
* scalb()
* signum()
* sino
* sinh0
* sqrt()
* subtractExact()
* tan()
* tanh()
* to Degrees()
* toIntExact()
* toRadians()
* ulp0

Java Output Methods

* Print()
* Prinftf()
* Println()

Java Arrays Methods

* Compare()
* equals()
* sort()
* fill()
* length()

Java ArrayList Methods

* add()
* addAll()
* clear()
* clone()
* contains
* ensureCapacity()
* forEach()
* get()
* indexOf()
* isEmpty()
* iterator()
* lastIndexOf()
* listIterator()
* remove()
* removeAll()
* removelf()
* replaceAll()
* retainAll()
* set()
* size()
* sort()
* spliterator()
* subList()
* toArray()
* trimToSize()

Java LinkedList Methods

* add()
* addAll()
* clear()
* clone()
* contains
* forEach()
* get()
* getFirst()
* getLast()
* indexOf()
* isEmpty()
* iterator()
* lastIndexOf()
* listIterator()
* remove()
* removeAll()
* removeFirst()
* removelf()
* removeLast()
* replaceAll()
* retainAll()
* set()
* size(
* sort()
* spliterator()
* subList()
* toArray()

Java HashMap Methods

* clear()
* clone()
* compute()
* computelfAbsent()
* computelfPresent()
* containsKey()
* containsValue()
* entrySet()
* forEach()
* get()
* getOrDefault()
* isEmpty()
* keySet()
* merge()
* put(
* putAll()
* putlfAbsent()
* remove()
* replace()
* replaceAll()
* size()
* values()

Java Scanner Methods

* close()
* delimiter()
* findInLine()
* findWithinHorizon()
* hasNext()
* hasNextBoolean()
* hasNextByte()
* hasNextDouble()
* hasNextFloat()
* hasNextInt()
* hasNextLine()
* hasNextLong()
* hasNextShort()
* locale()
* next()
* nextBoolean()
* nextByte()
* nextDouble()
* nextFloat()
* nextInt()
* nextLine()
* nextLong()
* nextShort()
* radix()
* reset()
* useDelimiter()
* useLocale()
* useRadix()

Java Iterator Methods

Java Errors & Exceptions

***Can you concisely identify which topics I should know and would be relevant to the project. (E.g. You should know all of Java Basics but … Or from Java Basics know, item 1, item 2…). If there is anything that you aren’t sure of clarify in a concise response.***

## Answer

For your **Basic File Encryption/Decryption** project in Java, here's a concise breakdown of the relevant topics from the list you provided:

**Java Basics**

You should know all of this, as it's foundational for understanding the syntax and writing code:

* **Java Syntax**
* **Java Comments**
* **Java Variables**
* **Java Data Types**
* **Java Operators**

**Java Strings**

Since the project involves text manipulation (for encryption and decryption), understanding how to handle strings is crucial:

* **Strings**
* **Concatenation**
* **Special characters** (for handling encrypted/decrypted output)

**Java If...Else**

Control flow is important for checking conditions (e.g., choosing encryption or decryption mode):

* **If**
* **Else**
* **Else if**

**Java Loops**

Loops will help in iterating over text and performing encryption/decryption:

* **While Loop**
* **For Loop**
* **For-each loop**

**Java Methods**

Methods are essential for structuring your encryption and decryption functionality:

* **Java Methods**
* **Java Method Parameters**
* **Return values**

**Java File Handling**

This is crucial for reading and writing to files:

* **Java Files**
* **Create/Write Files**
* **Read Files**

**Java User Input**

For taking input from users (e.g., file name, encryption key):

* **Scanner Methods**
* **nextLine()**
* **nextInt()**

**Java Exceptions**

Handling errors like missing files or incorrect input is essential:

* **Java Errors & Exceptions**

**Java String Methods (helpful for text manipulation, like trimming or replacing characters)**

* **substring()**
* **toLowerCase() / toUpperCase()**
* **trim()**

# Overview of the project

**Prompt**

Can you tell me how the project will work. First start with an overview of the project. Next, break down each step and give an example. Do not give the code solutions as I want to figure it out.

**Answer**

Overview of the Project

This project will involve creating a simple file encryption and decryption program using Java. The program will allow users to:

1. Select a file to encrypt or decrypt.
2. Specify a key (e.g., a number for a Caesar cipher).
3. Save the result to another file.
4. Optionally decrypt an encrypted file back to its original content using the same key.

The encryption method will be basic (e.g., shifting characters using a Caesar cipher), and the program will handle input and output via text files.

A screenshot of a computer error

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer error

Description automatically generated

A screenshot of a computer program

Description automatically generated

# How does shifting the characters work

A screenshot of a computer

Description automatically generated

# Answers

Your program effectively implements the basic encryption and decryption functionality using a Caesar cipher. However, there are several areas for improvement, both for addressing the issue you've mentioned and for enhancing the overall code quality, structure, and maintainability.

**Addressing the File Name Issue:**

If the user enters a wrong file name, your program terminates because it encounters a FileNotFoundException. You can solve this by using a do-while loop to repeatedly ask the user for a valid file name until a valid file is provided. Here's the updated part of the code:

String fileText = "";

File readObj = null;

do {

System.out.println("Enter the file name (with extension) you want to encrypt or decrypt:");

try {

String filePath = "C:\\Users\\JC\\Documents\\Programming Projects\\Basic-File-Encryption-Decryption\\files\\" + selectFileStorage.nextLine();

readObj = new File(filePath);

if (!readObj.exists() || !readObj.isFile()) {

System.out.println("File not found. Please enter a valid file name.");

continue;

}

Scanner myReader = new Scanner(readObj);

while (myReader.hasNextLine()) {

fileText += myReader.nextLine() + "\n"; // Append newline to preserve file structure

}

myReader.close();

} catch (Exception e) {

System.out.println("An error occurred. Please try again.");

readObj = null;

}

} while (readObj == null);

**Suggested Changes and Insights:**

1. **Encapsulation of Logic in Methods**:
   * Break down the main method into smaller methods to improve readability and maintainability.
   * For example, create separate methods for file reading, file writing, user input validation, etc.
2. **Error Handling**:
   * Use try-catch blocks judiciously. Avoid catching generic Exception unless necessary, and provide more specific handling for different exceptions.
3. **Use of Constants**:
   * Paths like C:\\Users\\JC\\Documents\\Programming Projects\\... should be stored in a constant or taken as input to make the program more flexible.
4. **Prevent Redundant Scanners**:
   * You have created multiple Scanner objects (encryptionPreference, selectFileStorage, etc.). A single Scanner object can be reused to avoid resource overhead.
5. **Improving the Encryption Logic**:
   * Consider expanding the logic to handle uppercase letters, digits, and special characters.
6. **File Not Found Case**:
   * You should explicitly check if the file exists before trying to read it, which can avoid unnecessary exceptions.
7. **Output File Names**:
   * Allow users to specify output file names instead of hardcoding them (encrypted.txt or decrypted.txt).
8. **Resource Management**:
   * Ensure Scanner and FileWriter objects are closed properly in a finally block or use a try-with-resources statement.

import java.io.\*;

import java.util.Scanner;

public class Encryption {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice = getEncryptionOption(scanner);

String inputFilePath = getFilePath(scanner, "Enter the file name to encrypt or decrypt:");

int shiftKey = getShiftKey(scanner);

String fileText = readFileContent(inputFilePath);

if (fileText == null || fileText.isEmpty()) {

System.out.println("Error: The file is empty or could not be read.");

return;

}

String result;

if (choice == 1) {

result = encrypt(fileText, shiftKey);

System.out.println("Encrypted content:\n" + result);

} else {

result = decrypt(fileText, shiftKey);

System.out.println("Decrypted content:\n" + result);

}

String outputFilePath = getFilePath(scanner, "Enter the name for the output file:");

writeFileContent(outputFilePath, result);

System.out.println("Operation completed successfully. Output saved to " + outputFilePath);

}

private static int getEncryptionOption(Scanner scanner) {

int option = 0;

while (option != 1 && option != 2) {

System.out.println("Press 1 for encryption or 2 for decryption:");

if (scanner.hasNextInt()) {

option = scanner.nextInt();

} else {

scanner.next(); // Clear invalid input

}

}

return option;

}

private static String getFilePath(Scanner scanner, String prompt) {

System.out.println(prompt);

return scanner.next();

}

private static int getShiftKey(Scanner scanner) {

System.out.println("Enter the shift key (positive integer):");

while (!scanner.hasNextInt()) {

scanner.next(); // Clear invalid input

System.out.println("Invalid input. Please enter a positive integer.");

}

return scanner.nextInt();

}

private static String readFileContent(String filePath) {

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

StringBuilder content = new StringBuilder();

String line;

while ((line = reader.readLine()) != null) {

content.append(line).append("\n");

}

return content.toString();

} catch (IOException e) {

System.out.println("Error reading file: " + e.getMessage());

return null;

}

}

private static void writeFileContent(String filePath, String content) {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {

writer.write(content);

} catch (IOException e) {

System.out.println("Error writing to file: " + e.getMessage());

}

}

public static String encrypt(String fileText, int shift) {

StringBuilder encryptedMessage = new StringBuilder();

for (char c : fileText.toCharArray()) {

if (Character.isLetter(c)) {

char base = Character.isLowerCase(c) ? 'a' : 'A';

c = (char) ((c - base + shift) % 26 + base);

}

encryptedMessage.append(c);

}

return encryptedMessage.toString();

}

public static String decrypt(String fileText, int shift) {

return encrypt(fileText, 26 - shift % 26); // Reuse encrypt for decryption

}

}

**Benefits of Refactoring:**

1. Enhanced readability and maintainability.
2. Better resource management with try-with-resources.
3. Modularized logic for reusability.
4. Handles edge cases like invalid inputs and empty files effectively.
5. Allows flexible file paths for both input and output.