**JAVA PROJECT 3: PROMPT**

# PROMPT start

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.

**RESPONSE**

Of course! I'd be happy to assist you in building a Java project step by step. Just let me know your idea or what kind of project you'd like to create, and we can start from there. I'll also clarify each step as we go. 😊

# PROMPT: Project context

Context

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.

}

Does this make sense. Just concisely summarise what I have said.

**PROMPT**

I already generated a plan. I plan to do this project next:

**Symmetric Encryption with a Key**  
*Difficulty: Moderate*

**Goal**: Implement a basic symmetric encryption algorithm (e.g., AES). The program should encrypt and decrypt text using a shared key provided by the user.

**Skills Covered**:

* + Cryptographic libraries (e.g., javax.crypto)
  + Working with byte arrays and strings
  + Exception handling (basic)

Now give me details of the project in this order: A summary, a deadline of the project (date), explain step by step what the project is and how it works with examples (do NOT give the code as I want to figure it out myself). The project should be expected to take 2 weeks.

## Answer

**Summary**

This project involves implementing a basic **symmetric encryption** algorithm (e.g., AES) in Java. The program will allow users to encrypt and decrypt a message using a shared key. This will teach you how to work with cryptographic libraries, handle byte arrays and strings, and manage exceptions.

**Deadline**

**Start Date**: January 5, 2025  
**End Date**: January 19, 2025

**Step-by-Step Explanation**

**Step 1: Understand Symmetric Encryption**

* **Concept**: Symmetric encryption uses the same key for both encryption and decryption.
* **How it works**: A plaintext message is transformed into ciphertext using a key. The same key is used to reverse the ciphertext back to plaintext.

**Example**:

* Plaintext: "Hello, World!"
* Key: "mySecretKey123"
* Ciphertext (encrypted): "aX34@!hsb" (looks random)
* Decrypted back to: "Hello, World!"

**Step 2: Design the Workflow**

1. **User Input**:
   * Take a plaintext message and a key from the user.
   * Ensure both inputs are valid (e.g., non-empty, correct length).
2. **Encryption**:
   * Use a library like javax.crypto to encrypt the message.
   * Convert the plaintext to a byte array and perform encryption.
   * Output the encrypted text (ciphertext) as a readable string (e.g., Base64 encoding).
3. **Decryption**:
   * Take the ciphertext and the same key as input.
   * Decode the ciphertext and decrypt it back into plaintext using the key.
4. **Error Handling**:
   * Handle invalid inputs, incorrect keys, or any cryptographic exceptions gracefully.

**Step 3: Example Walkthrough**

**Input**:

* Message: "Hello, Secure World!"
* Key: "password12345678" (AES requires keys of specific lengths, e.g., 16, 24, or 32 bytes).

**Encryption Process**:

1. Convert the message into bytes: "Hello, Secure World!" → [72, 101, ...].
2. Use the key to encrypt the byte array using AES.
3. Generate ciphertext: QWERTY@12345... (Base64 string).

**Decryption Process**:

1. Take the ciphertext and key as input.
2. Decode the ciphertext back into a byte array.
3. Use the same key to decrypt it into the original message: "Hello, Secure World!".

**Step 4: Testing and Validation**

* Test with different messages and keys to verify the program handles all cases.
* Test edge cases like an empty message or incorrect key.

Let me know if you need help as you progress!

# Prompt hw3schools

I am using w3schools to look at Java concepts. Here is all the concepts they cover. Note that dot points are a subheading of a heading. E.g. Under the Java Basics, you have as headings Java Syntax and Java Output. Subheadings under Java Output is “Print Text” and “Print Numbers”.

**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 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.