**JAVA PROJECT 3: PROMPT**

# 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.

**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

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: Password Hashing Simulator

Difficulty: Moderate

Goal: Build a program that simulates password hashing using a cryptographic hash function like SHA-256 (use Java’s built-in libraries for hashing).

o Skills Covered:

 Using external libraries (java.security package)

 Reading user input

 Understanding hash algorithms

**Answer**

Great choice! A **Password Hashing Simulator** is an excellent project to strengthen your understanding of cryptographic concepts and Java programming. Here’s a simple breakdown of how we can approach it:

**Steps:**

1. **Understand the Basics**:
   * Research SHA-256 and why hashing is used for passwords.
   * Learn about the java.security.MessageDigest class.
2. **Set Up Your Project**:
   * Create a new Java project.
   * Include comments to document your work as you go.
3. **Core Features**:
   * **User Input**: Prompt the user to enter a password.
   * **Hashing**: Use the MessageDigest class to hash the password with SHA-256.
   * **Output**: Display the original password and its hashed value.
4. **Optional Enhancements**:
   * Add a salt to make the hash more secure.
   * Simulate storing and verifying passwords.

Would you like me to create a project outline or write a starter code snippet for you?

**PROMPT**

Create a project outline and details of the project. The project should take 2 weeks. Also list the start date (today) and the due date (2 weeks from now) at the top under the summary. Also include examples outlining step by step hwo the project should work.

**Answer**

**Password Hashing Simulator Project**

**Start Date:** December 12, 2024  
**Due Date:** December 26, 2024

**Summary**

The Password Hashing Simulator project is designed to simulate how password hashing works using the SHA-256 cryptographic hash function. The project will involve reading user input, applying the hash algorithm, and displaying the hashed output. Additional features like adding salts and verifying passwords will provide further learning opportunities.

**Project Outline**

**Week 1: Understanding Basics and Core Implementation**

1. **Day 1-2: Research**
   * Study cryptographic hash functions (focus on SHA-256).
   * Understand Java's MessageDigest class from the java.security package.
2. **Day 3-4: Basic Setup**
   * Create a new Java project.
   * Set up a main class and prepare to take user input using Scanner.
3. **Day 5-7: Core Implementation**
   * Implement the password hashing functionality using MessageDigest.
   * Display both the original password and the hashed value in a readable format.
   * Test the program with different passwords.

**Week 2: Enhancements and Finalization**

1. **Day 8-10: Adding Salts**
   * Modify the program to generate a random salt for each password.
   * Hash the password + salt combination.
   * Store the salt and hashed password for later use.
2. **Day 11-12: Password Verification**
   * Implement a feature to verify a password by hashing the input and comparing it to a stored hash.
   * Test with correct and incorrect password inputs.
3. **Day 13: Documentation**
   * Write clear comments explaining your code.
   * Create a README file with project details, examples, and instructions.
4. **Day 14: Final Testing and Submission**
   * Test the program thoroughly for edge cases (e.g., empty passwords, special characters).
   * Ensure all requirements are met.
   * Submit the project.

**Detailed Project Flow (Step by Step)**

**Step 1: User Input**

* **Prompt:** "Enter a password to hash: "
* **Input Example:** mySecurePassword123

**Step 2: Generate Hash**

* Use MessageDigest to hash the input using SHA-256.
* Output Example:

Original Password: mySecurePassword123

Hashed Password: 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd2f88f76e8b066d0a8

**Step 3: Add Salt (Optional Enhancement)**

* Generate a random salt using SecureRandom.
* Combine the salt with the password before hashing.

**Output Example:**

Original Password: mySecurePassword123

Salt: XyZ12

Hashed Password: d9f9a9775dc7e3309f52e4af6eec8e9d88b0833d8fd324038edf1e95c1501021

**Step 4: Verify Password (Optional Enhancement)**

* Prompt the user to enter the password again for verification.
* Hash the input with the stored salt and compare it with the stored hash.

**Output Examples:**

Correct Password:

Password Verified Successfully!

Incorrect Password:

Password Verification Failed.

**Tools and Libraries**

* **IDE**: IntelliJ IDEA, Eclipse, or any text editor with Java support.
* **Java Libraries**:
  + java.util.Scanner (for user input).
  + java.security.MessageDigest (for hashing).

# Prompt

Explain how to use MessageDigest first in concise begineer friendly language. Don't give me the starter code as I want to figure it out myself. Next just tell me what you plan to suggest for starter day code.

**How to Use MessageDigest (Beginner-Friendly)**

1. **What is MessageDigest?**  
   It's a Java class in the java.security package that helps you create hash values, like SHA-256, for any data (e.g., passwords).
2. **Steps to Use It**:
   * **Import**: Include java.security.MessageDigest at the top of your program.
   * **Get Instance**: Use MessageDigest.getInstance("SHA-256") to create a hashing object.
   * **Update Data**: Convert your input (e.g., password) into bytes using .getBytes() and feed it to the hashing object.
   * **Generate Hash**: Call .digest() on the hashing object. This will return the hash as a byte array.
   * **Convert to Hex**: Use a loop or helper method to convert the byte array into a readable hexadecimal format.
3. **Key Points**:
   * SHA-256 is secure and widely used.
   * Hashes are one-way: you can’t reverse a hash to get the original data.

**Starter Code Plan for Days 1-3**

* **Day 1**: Create a project and set up a Java file with the main method. Import the necessary libraries and write a placeholder message like System.out.println("Password Hashing Simulator");.
* **Day 2**: Add code to prompt the user to input a password using Scanner. Print the entered password to confirm it's being read.
* **Day 3**: Use MessageDigest to hash the password and print the raw byte array result (without converting to hex yet).

Would you like a high-level explanation for converting the hash to hex when you're ready?

# Prompt: hw3school

No I do not need a high level explanation for ocnverting a has to hex. I will try to figure that out myself. Now answer the following: 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

**Relevant Topics for the Password Hashing Simulator**

**1. Java Basics**

* **Essential**:
  + **Java Syntax**
  + **Java Output**: Understand Print Text and Print Numbers.
  + **Java Comments**: Useful for documenting your code.
  + **Java Variables**: Including multiple variables.
  + **Java Data Types**: Numbers, Booleans, Characters, and Non-primitive types.
  + **Java Type Casting**: When dealing with type conversions.
  + **Java Strings**: Including concatenation and special characters.

**2. Java Operators**

* **Only Basic Knowledge**: Arithmetic and comparison operators will suffice.

**3. Java Math**

* **Optional**: Not directly relevant unless you’re planning on experimenting with randomness or advanced math.

**4. Java If...Else & Loops**

* **Essential**:
  + Basic **if/else** and **switch** statements.
  + **For** and **while** loops to repeat hashing experiments.

**5. Java Methods**

* **Critical**:
  + Creating and using methods (Parameters and Return values).
  + **Java Method Overloading** is optional.

**6. Java Classes & Objects**

* **Essential**: Understand **Classes/Objects**, **Attributes**, **Constructors**, and **Methods**.

**7. Java User Input**

* **Critical**: Learn how to use Scanner to take input from users.

**8. Java Exceptions**

* **Important**: Learn basic try-catch to handle errors (e.g., invalid hash algorithms).

**9. Java Packages**

* **Critical**: Understand how to import libraries like java.security.MessageDigest.

**10. Java String Methods**

* **Essential**:
  + length(), substring(), toLowerCase(), toUpperCase(), and trim().
  + equals() and equalsIgnoreCase() for string comparisons.

**11. Java Advanced Topics**

* **Optional**: Topics like Polymorphism, Abstraction, and Threads are not needed.

**12. Java File Handling**

* **Optional**: Only relevant if you want to store hash values in files.

**13. Java Scanner Methods**

* **Essential**: Learn next(), nextLine(), and hasNext() to handle input.

**14. Java HashMap/ArrayList**

* **Optional**: Only if you want to store multiple passwords and hashes for testing.

If you're unsure about any specific concept while working, feel free to clarify with examples or code snippets!

# Prompt: Project clarification

I am confused on the project. Is the project just gettng the user to enter their password and then encoding it in SHA-156. I can then if I I have time research salt and storing the user password?

## Answer

Yes, you're correct in your understanding of the project's core goal:

**Core Objective**

1. **Take User Input**: Allow the user to input their password.
2. **Hash the Password**: Use SHA-256 (not SHA-156; it doesn't exist) to encode the password securely.
3. **Output the Hash**: Display the resulting hash as a hexadecimal string.

**Optional Extensions (if time permits)**

1. **Implement Salting**:
   * Generate a random "salt" (a unique value) for each password.
   * Combine the password and salt before hashing to strengthen security.
2. **Store User Credentials**:
   * Save the hashed password (and salt, if used) in a file or data structure for later verification.
3. **Password Verification**:
   * Ask the user to enter their password again.
   * Re-hash the entered password (with the salt, if applicable) and compare it to the stored hash to verify if it matches.

**Why Salt and Storage?**

* **Salting** makes it much harder for attackers to use precomputed hash tables (like rainbow tables) to crack passwords.
* **Storing** hashed passwords (not the plaintext password) is a real-world approach for user authentication systems.

If you're new to this, **focus on hashing the password with SHA-256 first**. Once you're comfortable, you can explore salting and storage. This way, you can complete the core project while leaving room for extra features if time allows!