# Java Program: Data Manager with Encryption

## Group Members:

1. Member 1 Name

2. Member 2 Name

3. Member 3 Name

## Introduction

This report details the structure, functionality, and design of a Java-based Data Manager program. The program encrypts data using XOR encryption, stores it securely, and allows for decryption with the correct key. The application adheres to Object-Oriented Programming (OOP) principles, emphasizing modularity, encapsulation, and abstraction.

## Modularization

The program is divided into four primary classes, each with a specific responsibility. This modular design ensures code reusability, maintainability, and scalability.

### Class Responsibilities and Methods

|  |  |  |
| --- | --- | --- |
| Class | Methods | Responsibilities |
| Main | main() | Program entry point. |
| MenuManager | displayMenu(), handleInputData(), handleViewDatabase(), clearScreen(), pause() | Manages user interaction and program flow. |
| Database | addEntry(), getFormattedEncryptedData(), decryptEntry(), isFull(), isEmpty() | Handles storage, retrieval, and decryption of data. |
| EncryptionDecryption | encrypt(), decrypt(), process(), getKey() | Encrypts and decrypts data using XOR logic. |

## Textual Representation Diagram

The following diagram illustrates the relationships between the classes in the program.

Main  
 |  
 v  
MenuManager  
 |  
 +-----------------------------+  
 | |  
 v v  
Database EncryptionDecryption  
 |  
 +-- Stores encrypted data +-- Handles encryption and decryption  
 +-- Associates encryption +-- Encapsulates XOR logic  
 keys with entries

## Sample Workflow

1. \*\*Add Entry\*\*:   
- User inputs data and key.  
- Data is encrypted via `EncryptionDecryption` and stored in `Database`.  
- Output: Data has been encrypted and stored.

2. \*\*View Encrypted Data\*\*:   
- User views all stored encrypted data in the format:  
 The encrypted data of "Index number 1" index is "EncryptedValue1".  
- User selects an entry and provides a decryption key.

3. \*\*Decrypt Data\*\*:   
- If the key matches, decrypted data is shown.  
- If the key is incorrect, an error message is displayed.

## Conclusion

This program demonstrates the practical application of Object-Oriented Programming principles. Its modular design makes it robust, scalable, and maintainable, while the encryption mechanism ensures data security. By clearly separating responsibilities into distinct classes, the program adheres to industry best practices.

## Summary of OOP Pillars

### 1. Encapsulation

Encapsulation involves bundling data and methods into a single unit and restricting direct access to them. In the program, the `Database` and `EncryptionDecryption` classes encapsulate their internal data and expose only necessary methods like `addEntry()` and `encrypt()`. This ensures that sensitive data, such as encryption keys, is not directly accessible to users.

### 2. Abstraction

Abstraction hides the complex implementation details from the user and provides a simplified interface. For example, users interact with the program through the `MenuManager` class without needing to understand the encryption process handled by `EncryptionDecryption`. This makes the program user-friendly and secure.

### 3. Inheritance

Inheritance allows classes to inherit properties and methods from other classes, enabling code reuse and extensibility. While the current program does not explicitly use inheritance, its modular design supports future extensions. For instance, the `EncryptionDecryption` class could serve as a base class for more advanced encryption methods.

### 4. Polymorphism

Polymorphism enables the same interface to be used for different implementations. The program can be extended to include multiple encryption strategies by leveraging polymorphism. For instance, the `encrypt()` method could be implemented differently for various encryption algorithms while maintaining the same interface.