ООП Password Manager представяне

Част 1 – Шифрите

Йерархията започва от чистия абстрактен клас Cipher. Той изисква пет метода – encrypt, decrypt, name, writeConfig и readConfig. С тези методи подчертавам какво се изисква от всеки един от шифърите а именно: всеки шифър трябва да може да кодира и декодира текст, да се представя еднозначно чрез кратък идентификатор и да знае как да запише и възстанови собствената си конфигурация във файл-поток.

CaesarCipher

Запазва едно цяло число shift; при шифриране от позицията на всеки символ в ASCII-диапазона се изважда shift, а резултатът се модифицира по модул 95, за да „увие“ около края на таблицата.

Дешифрирането е огледален процес с прибавяне на shift.

writeConfig записва shift като сурови 4 байта; readConfig ги възстановява, така че обектът се самоконфигурира при отваряне на файл с пароли.

TextCodeCipher

При построяване приема път до текстов файл („референтен текст“). Файлът се сканира линейно; първата поява на всеки допустим символ получава уникален пореден номер. Получават се две таблици: \_encode[символ] → индекс и \_decode[индекс] → символ.

Шифрирането заменя всеки знак с неговия индекс, изведен като поредица от цифри, разделени с интервали.

При декодиране индексите се събират от потока до интервал, валидират се и се превръщат обратно чрез \_decode.

writeConfig извежда броя елементи и самата \_decode-последователност, което позволява алгоритъмът да работи без повторно зареждане на файла; readConfig прави обратното и регенерира \_encode.

HillCipher

Ключът е квадратна матрица mod 95, прочетена от външен файл – първият елемент е размерът n, следват n² цели числа.

Шифрирането групира текста в вектори от n символа (допълва с интервали), умножава ги с матрицата, намалява резултата mod 95 и превръща числата обратно в символи.

За дешифриране се изчислява инверсираната матрица \_invKey чрез целочислен Гаус-Жордан mod 95. Тук се използва AI-генерирана имплементация, тъй като ръчното разработване и проверка на стабилен инвертор би изместило фокуса от архитектурата към числени детайли.

writeConfig записва n и всички елементи на матрицата; readConfig ги възстановява и пресмята \_invKey, така че при open не са нужни допълнителни аргументи.

**Част 2 – Плъгин-архитектурата (CipherCreator, AutoCreator, CipherFactory)**

Архитектурата разчита на класически Factory модел, разширен до plug-in стил, така че добавянето на нов шифър изисква един-единствен нов .cpp/.h чифт и никакви изменения на съществуващия код.

CipherCreator

Декларативен „визитник“: пази идентификатор на шифъра и две виртуални функции – createFromArgs (когато потребителят подава параметри от конзолата) и createFromStream (когато файлът вече съдържа конфигурация). Самият конструктор се регистрира във CipherFactory.

AutoCreator<T>

Универсален шаблон, който реализира двете фабрични функции по подразбиране:

Аргументи → извиква статичната T::makeFromArgs – авторът на шифъра описва един-единствен публичен метод за парсване.

Поток → създава празен T() (изисква дефолтен конструктор), след което подава потока към readConfig.

По този начин няма нужда всеки шифър да има собствен малък клас-създател; отпада дублиран код, регистрацията е едноредова (static AutoCreator<T> \_;). Единственият компромис е изискването за дефолтен конструктор.

CipherFactory

Сингълтън, който държи вектор от всички регистрирани CipherCreator екземпляри. При createCipherFromArgs или createCipherFromStream фабриката обхожда списъка, съпоставя getName() с идентификатора, и делегира създаването към съответния CipherCreator. Целият Command-driven слой вижда само един метод – подава име на шифъра плюс аргументи или поток, а в замяна получава готов обект Cipher.

Част 3 – PasswordManager и CommandHandler

PasswordManager държи логиката за файловия формат, криптирането и всички CRUD-операции върху запомнените пароли. Файлът започва със:

криптиран <filePass> – глобалната парола за достъп;

името на шифъра;

конфигурацията, изведена от writeConfig;

броя записани пароли и самите записи – всеки със собствена дължина и съдържание.

Разделянето на мениджъра от CommandHandler позволява конзолният интерфейс да бъде сменен с GUI или уеб услуга без докосване до защитния слой.

PasswordManager::create – приема име на файл, нов шифър (получен вече от фабриката) и <filePass>, занулява списъка \_entries и извиква saveToFile.

open – прочита първите три полета, създава подходящия шифър през фабриката, декриптира <filePass> и при успех зарежда останалите записи.

savePassword – добавя свеж запис и извиква оптимизираното appendEntryToFile, което:

позиционира се на запомнената позиция \_countPos, обновява броя,

връща се в края и извежда само новите байтове.

Така файлът не се пренаписва изцяло, а остава консистентен с минимален I/O.

CommandHandler е тънка обвивка над std::cin: разбива командния ред по интервали, валидира брой аргументи и извиква съответния метод на мениджъра. Разделението дава две ключови предимства – яснота (един клас обработва I/O, друг – бизнес логика) и лесна подмяна на потребителския интерфейс.