Домашня робота:

Дочитайте, будь ласка, до кінця, там є загальні вимоги до всих варіантів домашнього завдання.

Реалізуйте ОДИН з таких патернів проектування по специфікації:

1. **Фабричний метод**.
   Реалізуйте консольний застосунок, який:
   - отримує повний шлях до папки у вхідному параметрі
   - збирає список файлів і папок, які знаходяться в цій папці (можете реалізувати рекурсивно, якщо маєте час та натхнення, або нерекурсивно), також збирає їхні властивості такі як розмір, час створення, час модифікації, час останнього доступу, тип, тільки-для-читання, архівний та інше, не менш як 3 властивості (за Вашим вибором).
   - друкує «звіт» про вміст папки в таких форматах: txt, csv, xml, json (оберіть не менше як два). Під словом «формат» мається на увазі не тільки розширення файлу, а і його вміст.
   - використовуйте фабричний метод для друку звіту

2. **Спостерігач (Observer).**
   Реалізуйте консольний застосунок, який:
   - Друкує “hello world” (або будь-який інший текст) в консолі.
   - і спостерігає за рухом миші:
   Коли курсор заходить в область консольного вікна, колір тексту в консолі змінюється, а поточний час записується в лог файл з тегом «Enter».
   Коли курсор полишає область консольного вікна, колір тексту в консолі змінюється на оригінальний і поточний час записується у лог з тегом “Leave”.
   Ви можете захардкодити шлях до лог файлу або розмістити його в той же директорії що й exe файл.
   Використайте патерн Спостерігач.
3. **Адаптер**
   1. Напишіть клас, який реалізує пошук по файловій системі, тобто отримує шлях, де починати пошук і шукає файли і папки всередині цього шляху. Має публічний метод GetNext. Він приймає строку, а далі перевіряє імена знайдених файлів і папок на входження цієї строки. Відповідно якщо в імені такої підстроки нема, то файл/папка не потрапляють у видачу пошуку, а ваш клас шукає і перевіряє наступний файл/папку. Якщо ж файл/папку знайдено, то додатково повертаємо час створення файлу, розмір і, звісно, назву, а також шлях, щоб точно ідентифікувати його.

2. Не змінюйте цей клас. Тепер вважаємо це легасі кодом, якій ми не можемо змінити і перекомпілювати.
3. Поміняємо умови. Тепер вхідна строка нехай буде не юнікодна, а UTF-8 (чи навпаки, якщо у вас до того було інше кодування), на виході час буде не локальний, а UTC, чи навпаки локальний + таймзона, якщо до того був UTC, розмір буде не в байтах, а в кілобайтах, відповідно назва файлу і шлях теж в іншому кодуванні. Зміна тих всих параметрів повинна відображатися на назвах змінних, методів і т.д.
4. Реалізуйте нові умови за допомогою патерна Адаптер.
5. Продемонструйте використання оригінальної системи пошуку, а потім нового адаптеру. Мається на увазі зробіть код, який використовує і те і інше. А результати можна подати окремим файлом чи скріншотом. Писати код для зберігання результатів у файл не обов'язково.

Зауваження до всих варіантів домашнього завдання:

Ви можете використовувати стандартну бібліотеку STL та/або boost

Всі завдання мають задовольняти таким вимогам:

1. Використовуйте розумні вказівники для управління пам'яттю. Можете використовувати самописні гварди (guards) або рішення з stl/boost для інших видів ресурсів.
2. Використовуйте контейнери з stl: vector, map, list і т.д., за потреби.
3. Використовуйте винятки (exceptions) для обробки помилок. Забезпечте посилену (strong) гарантію безпеки і не забудьте про best practices.
4. Продумайте архітектуру застосунку: інтерфейси, відповідальність класів (не забудьте про принцип єдиної відповідальності). Оскільки ми обговорювали ООП і патерни проектування, архітектура є важливою частиною домашньої роботи. Ви маєте показати наскільки Ви засвоїли як працюють патерни і ООП.
5. Розподіліть класи по файлах. Це також важлива частина декомпозиції. Можна розмістити декілька класів в один файл, якщо це відповідає логіці застосунку, але не мішайте усе в main.cpp.

Принципи оцінювання:
Кожен матиме максимально балів (100) на початку. Кожна помилка коштуватиме балів.
Що вважається помилкою:
- Ви не чітко розумієте як працює обраний патерн.
- Неправильна обробка помилок, або її відсутність. Кидайте винятки (exceptions) коли ваша логіка вимагає виняткової ситуації. У більшості випадків Вам достатньо std::runtime_error. Не забувайте десь зловити ті винятки і показати помилки користувачу. Пам'ятайте що stl/boost функції також можуть кидати винятки, але деякі функції (особливо Windows API) можуть натомість вертати статуси, тож читайте документацію по функціях і обробляйте помилки відповідно.

- Проблеми з управлінням ресурсами: витік ресурсів, неправильне використання розумних вказівників. Щодо non-memory ресурсів, читайте документацію про те як їх правильно звільняти.
- Неправильна декомпозиція, змішана відповідальність, недостатня інкапсуляція, зайві поля або методи в класах.
- Нереалізовані вимоги (з мінімального списку)
- Незрозумілі назви змінних, методів, файлів в т.д. Наприклад ex замість expectedResult. Але звісно що нормально використати локальну ex у блоку catch, або змінну i у циклі for.