Problem Solving (CPP23)

هذا البرنامج التدريبي مُصاغ بعناية لتمكين المتدربين من تطوير قدراتهم الفكرية على غرار المبرمجين المحترفين، والتعاون بكفاءة ضمن فريق محترف في شركة "جمال تك" أو أي مؤسسة متعددة الجنسيات أخرى. نظرًا لأهمية اللغة الإنجليزية في بيئة العمل العالمية، يتم تقديم المحتوى التدريبي بالإنجليزية. لا يشترط إتقان اللغة بشكل كامل، لكن من الضروري امتلاك القدرة الكافية لفهم المتطلبات وتنفيذها بشكل فعّال. يُمكن للمتدربين استخدام مترجم جوجل أو الاستعانة بـ "شات جي بي تي" للتغلب على أية عقبات لغوية، المهم هو الفهم الدقيق للمطلوب وتحقيقه بنجاح.

لتعظيم الاستفادة من التدريب، يُنصح بمحاولة حل التمارين بشكل مستقل لمدة ساعة واحدة على الأقل قبل الرجوع إلى الحل المرفق في نهاية الملف.

قد يتضمن الحل كودًا برمجيًا غير مفسر بعد، والغرض من ذلك هو تشجيعك على محاولة فهم الأكواد البرمجية الجديدة التي لم تتعرض لها من قبل. هذه المهارة ضرورية في سوق العمل، حيث تتطور لغات البرمجة باستمرار ويظهر كل يوم لغات جديدة. ستواجه دائمًا أكوادًا لم تدرسها من قبل، ومن المهم أن تكون قادرًا على فهمها بنفسك دون الحاجة إلى دراسة مسبقة. يمكنك الاستعانة بمحرك البحث جوجل، أو استخدام ChatGPT، أو حتى اللجوء لأصدقائك للمساعدة. الهدف الأساسي هو أن تصل إلى فهم معنى كل كود بأي طريقة ممكنة لتتمكن من إيجاد موقعك في سوق العمل.

إن وجود كود برمجي غير مفسر يشكل تحديًا يتوجب عليك إيجاد حل له. هذا النوع من التدريبات يعد جزءًا أساسيًا من تدريبات 'Problem Solving'، التي تهدف إلى تمكينك من أداء عملك بفاعلية بغض النظر عن التحديات والعقبات. هذه القدرة على حل المشكلات هي ما يتمتع به العاملون في 'جمال تك'، ومن الضروري أن تطور في نفسك هذه المهارة لتصبح عضوًا فعالًا في فريق عمل 'جمال تك'.

Gammal Tech's Design Pattern Decoder

Background

Gammal Tech, a software development company renowned for its innovative approach and state-of-the-art facilities, is organizing a workshop for its interns. The workshop's highlight is understanding and implementing design patterns, a cornerstone of efficient software design. Gammal Tech aims to demonstrate its expertise in this domain by presenting a real-world problem that simplifies complex system designs using a particular design pattern.

Problem Statement

As part of the training, Gammal Tech has a system that generates various software design scenarios. Each scenario describes a software design problem that needs a specific design pattern to solve. Your task is to create a program that reads these scenarios and identifies the most suitable design pattern.

The design patterns to consider are:

Singleton
Observer
Factory Method
Strategy
Decorator

Each scenario will be described in a single line, containing certain keywords that are uniquely associated with one of the design patterns above.

Input Format

- The first line contains an integer `N`, the number of scenarios.
- The next `N` lines each contain a string `S`, describing a scenario.

Output Format

For each scenario, output a single line containing the name of the design pattern that best fits the scenario.

Constraints

- `1 ≤ N ≤ 100`
- `1 ≤ |S| ≤ 1000`, where `|S|` is the length of the string.

Sample Input:

```
3
"Need a single configuration object for our application"
"Multiple views need to update when the data model changes"
"Creating objects without specifying the exact class of object that will
be created"
```

Sample Output:

```
Singleton
Observer
Factory Method
```

Explanation

- The first scenario suggests a need for a single shared resource, which is a typical use case for the Singleton pattern.
- The second scenario deals with multiple entities reacting to a change, aligning with the Observer pattern.
- The third scenario describes creating objects where the exact class isn't known, fitting the Factory Method pattern.

لتعظيم الاستفادة من التدريب، يُنصح بمحاولة حل التمرين بشكل مستقل لمدة ساعة واحدة على الأقل قبل الرجوع إلى الحل المرفق

C++ Programming Solution:

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

string identifyPattern(const string& scenario) {
    if (scenario.find("single configuration object") != string::npos) {
        return "Singleton";
    } else if (scenario.find("update when the data model changes") !=
string::npos) {
        return "Observer";
    } else if (scenario.find("without specifying the exact class of
object") != string::npos) {
        return "Factory Method";
    } else if (scenario.find("strategies for different operations") !=
string::npos) {
        return "Strategy";
    } else if (scenario.find("add responsibilities to objects
dynamically") != string::npos) {
        return "Decorator";
    }
    return "Unknown Pattern";
}

int main() {
    int N;
    cin >> N;
    cin.ignore();

    vector<string> scenarios(N);
    for (int i = 0; i < N; ++i) {
        getline(cin, scenarios[i]);
    }

    for (const auto& scenario : scenarios) {
        cout << identifyPattern(scenario) << endl;
    }

    return 0;
}
```