Problem Solving (CPP25)

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 Educator

Background

Gammal Tech, a vanguard in the software development industry, renowned for its trailblazing technologies and forward-thinking work culture, has embarked on a new educational initiative. They are developing a C++ program designed to teach budding programmers about software design patterns. Your task is to contribute to this initiative by developing a section of the program.

Problem Statement

You are to create a function in C++ that educates the user about a specific design pattern. Choose any well-known design pattern (like Singleton, Factory, Observer, etc.) and implement it in a way that not only demonstrates the pattern but also explains its components and usage through interactive examples.

Your program should first ask the user to select a design pattern from a list. Then, it should display a brief description of the chosen pattern, implement it, and finally, show a practical example of its use in software design.

Input Format

- A single integer `N` - the number of queries.
- `N` lines follow, each containing a string which is the name of a design pattern the user wants to learn about.

Output Format

For each query:

- A brief description of the chosen design pattern.
- The implementation of the design pattern in C++.
- An example showing the practical use of the design pattern.

Constraints

- `1 ≤ N ≤ 5`
- Each string is a valid design pattern name.

Sample Input:

```
2
Singleton
Factory
```

Sample Output:

```
Singleton Pattern:
Description: The Singleton pattern ensures that a class has only one
instance and provides a global point of access to it.
Implementation: [C++ code of Singleton pattern]
Example: [Example of Singleton pattern usage]

Factory Pattern:
Description: The Factory pattern defines an interface for creating an
object but lets subclasses alter the type of objects that will be
created.
Implementation: [C++ code of Factory pattern]
Example: [Example of Factory pattern usage]
```

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 <map>
#include <memory>

// Forward declaration of DesignPattern classes
class DesignPattern;
class SingletonPattern;
class FactoryPattern;

// Factory for creating instances of different DesignPattern types
class DesignPatternFactory {
public:
    static std::shared_ptr<DesignPattern> createPattern(const
std::string& patternName);
};

// Base class for design patterns
class DesignPattern {
public:
    virtual void describe() = 0;
    virtual void implement() = 0;
};

// Singleton Pattern class
class SingletonPattern : public DesignPattern {
public:
    void describe() override {
        std::cout << "Singleton Pattern:\n"
                  << "Description: The Singleton pattern ensures that a
class has only one instance and provides a global point of access to
it.\n";
    }

    void implement() override {
        std::cout << "Implementation: [C++ code of Singleton pattern]\n"
                  << "Example: [Example of Singleton pattern usage]\n";
    }
};
```

```cpp
// Factory Pattern class
class FactoryPattern : public DesignPattern {
public:
    void describe() override {
        std::cout << "Factory Pattern:\n"
                  << "Description: The Factory pattern defines an
interface for creating an object but lets subclasses alter the type of
objects that will be created.\n";
    }

    void implement() override {
        std::cout << "Implementation: [C++ code of Factory pattern]\n"
                  << "Example: [Example of Factory pattern usage]\n";
    }
};

// Implementation of DesignPatternFactory
std::shared_ptr<DesignPattern> DesignPatternFactory::createPattern(const
std::string& patternName) {
    if (patternName == "Singleton") {
        return std::make_shared<SingletonPattern>();
    } else if (patternName == "Factory") {
        return std::make_shared<FactoryPattern>();
    }
    return nullptr;
}

int main() {
    int N;
    std::cin >> N;
    std::string patternName;

    for (int i = 0; i < N; ++i) {
        std::cin >> patternName;
        auto pattern = DesignPatternFactory::createPattern(patternName);
        if (pattern) {
            pattern->describe();
            pattern->implement();
        } else {
            std::cout << "Unknown Design Pattern: " << patternName <<
std::endl;
        }
    }
    return 0;
}
```