Project Documentation

Your Name

November 14, 2023

Contents

1 EnvVar Class

1.1 Source Code

```hpp
// EnvVar.hpp

#ifndef ENVVAR_HPP
#define ENVVAR_HPP

#include <string>
#include <optional>
#ifdef THREAD_SAFE
#include <mutex>
#endif

class EnvVar {
public:
    explicit EnvVar(const std::string& name);

    std::string get() const;
    bool set(const std::string& value) const;
    void store();
    bool restore() const;

private:
    std::string varName;
    std::optional<std::string> storedValue;
#ifdef THREAD_SAFE
    static std::mutex mtx;  // Mutex for thread safety
#endif
};

#endif // ENVVAR_HPP
```

```cpp
// EnvVar.cpp

#include "EnvVar.hpp"
#include "Logger.hpp"
#include <cstdlib>
#ifdef THREAD_SAFE
#include <mutex>
std::mutex EnvVar::mtx;  // Define the static mutex
#endif

EnvVar::EnvVar(const std::string& name) : varName(name) {}

std::string EnvVar::get() const {
#ifdef THREAD_SAFE
```

```
    std::lock_guard<std::mutex> lock(mtx);  // Lock the mutex
#endif
    const char* value = std::getenv(varName.c_str());
    return (value != nullptr) ? std::string(value) : std::string();
}

bool EnvVar::set(const std::string& value) const {
#ifdef THREAD_SAFE
    std::lock_guard<std::mutex> lock(mtx);  // Lock the mutex
#endif
    //Logger::getInstance().log("Setting environment variable: " + varName + " to " + value, "EnvVar::set", Logger::Severity::Info);
    return setenv(varName.c_str(), value.c_str(), 1) == 0;
}

void EnvVar::store() {
#ifdef THREAD_SAFE
    std::lock_guard<std::mutex> lock(mtx);  // Lock the mutex
#endif
    storedValue = get();
}

bool EnvVar::restore() const {
#ifdef THREAD_SAFE
    std::lock_guard<std::mutex> lock(mtx);  // Lock the mutex
#endif
    if (storedValue.has_value()) {
        return set(storedValue.value());
    }
    return false;
}
```

2 ConfigManager Class

2.1 Source Code

```
// ConfigManager.hpp

#ifndef CONFIGMANAGER_HPP
#define CONFIGMANAGER_HPP

#include <iostream>
```

```cpp
#include <string>
#include <mutex>
#include <nlohmann/json.hpp>

class ConfigManager {
public:
    explicit ConfigManager(const std::string& configFilePath);
    ~ConfigManager();

    template<typename T>
    T get(const std::string& key) const;

    template<typename T>
    void set(const std::string& key, const T& value);

    void sync();

#ifdef THREAD_SAFE
    static std::mutex mtx;  // Mutex for thread safety
#endif

private:
    nlohmann::json config;
    std::string filePath;
    const nlohmann::json& getRefToValue(const std::string& key, bool forRead) const;
    nlohmann::json& getRefToValue(const std::string& key);
};

#endif // CONFIGMANAGER_HPP
```

```cpp
// ConfigManager.cpp

#include "ConfigManager.hpp"
#include "Logger.hpp"
#include <iostream>
#include <fstream>
#include <sstream>

#ifdef THREAD_SAFE
std::mutex ConfigManager::mtx;
#endif

ConfigManager::ConfigManager(const std::string& configFilePath) : filePath(configFilePath) {
  std::cerr << "Entering Constructor for ConfigManager" << std::endl;
    std::ifstream file(filePath);
```

```
    if (file) {
        try {
          std::cerr << "file found" << std::endl;
            file >> config;
        } catch (const nlohmann::json::parse_error& e) {
            Logger::getInstance().log("JSON parsing error: "
                + std::string(e.what()), "ConfigManager::
                ConfigManager", Logger::Severity::Error);
            std::cerr << "Configuration loading error. Check
                log file for details." << std::endl;
            config = nlohmann::json::object(); // Ensure
                config is a valid JSON object
        }
    } else {
        Logger::getInstance().log("Config file not found: "
            + filePath, "ConfigManager::ConfigManager",
            Logger::Severity::Warning);
        std::cerr << "Configuration file missing. A new one
            will be created." << std::endl;
        config = nlohmann::json::object(); // Initialize
            config as an empty object
    }

    // Additional check to ensure config is not null
    if (config.is_null()) {
        config = nlohmann::json::object();
    }
}

ConfigManager::~ConfigManager() {
    sync();
}

template<typename T>
T ConfigManager::get(const std::string& key) const {
#ifdef THREAD_SAFE
    std::lock_guard<std::mutex> lock(mtx);
#endif
    try {
        const nlohmann::json& ref = getRefToValue(key, true)
            ;
        return ref.get<T>();
    } catch (const nlohmann::json::out_of_range& e) {
        // Handle the case where the key does not exist
        Logger::getInstance().log("Key not found in
            configuration: " + key, "ConfigManager::get",
            Logger::Severity::Warning);
        throw std::runtime_error("Configuration key not
            found: " + key);
    } catch (const nlohmann::json::exception& e) {
```

```
54          // Handle other JSON exceptions
55          Logger::getInstance().log("Error accessing key '" +
                key + "': " + e.what(), "ConfigManager::get",
                Logger::Severity::Error);
56          throw;
57      }
58  }
59
60  template<typename T>
61  void ConfigManager::set(const std::string& key, const T&
        value) {
62  #ifdef THREAD_SAFE
63      std::lock_guard<std::mutex> lock(mtx);
64  #endif
65      nlohmann::json& ref = getRefToValue(key); // Use non-
            const ref
66      ref = value;
67  }
68
69  void ConfigManager::sync() {
70  #ifdef THREAD_SAFE
71      std::lock_guard<std::mutex> lock(mtx);
72  #endif
73      std::ofstream file(filePath);
74      if (file) {
75          file << config.dump(4);  // Save the JSON in a
                pretty format
76      }
77  }
78
79  const nlohmann::json& ConfigManager::getRefToValue(const std
        ::string& key, bool forRead) const {
80      const nlohmann::json* j = &config;
81      std::istringstream iss(key);
82      std::string token;
83      while (std::getline(iss, token, '.')) {
84          j = &((*j).at(token));
85      }
86      return *j;
87  }
88
89  nlohmann::json& ConfigManager::getRefToValue(const std::
        string& key) {
90      nlohmann::json* j = &config;
91      std::istringstream iss(key);
92      std::string token;
93      while (std::getline(iss, token, '.')) {
94          j = &((*j)[token]);
95      }
96      return *j;
```

```
}

// Explicit template instantiation
template int ConfigManager::get<int>(const std::string& key) const;
template std::string ConfigManager::get<std::string>(const std::string& key) const;
template void ConfigManager::set<int>(const std::string& key, const int& value);
template void ConfigManager::set<std::string>(const std::string& key, const std::string& value);
```

3 Logger Class

3.1 Source Code

```
// Logger.hpp

#ifndef LOGGER_HPP
#define LOGGER_HPP

#include <string>
#include <fstream>
#include <mutex>

class Logger {
public:
    enum class Severity {
        Info,
        Warning,
        Error
    };

    static Logger& getInstance();
    void log(const std::string& message, const std::string& location, Severity severity);

private:
    std::ofstream logFile;
    std::mutex mtx;

    Logger(); // Private constructor for Singleton pattern
    ~Logger();
    Logger(const Logger&) = delete;
    Logger& operator=(const Logger&) = delete;

    std::string severityToString(Severity severity);
};
```

```

#endif // LOGGER_HPP
```

```cpp
// Logger.cpp

#include "Logger.hpp"
#include "EnvVar.hpp"
#include <iostream>
#include <fstream>
#include <sstream>
#include <chrono>
#include <iomanip>

Logger::Logger() {
    std::string logPath = "testing.log"; // Default log file name

    // Use std::getenv directly to avoid dependency on EnvVar
    const char* configPath = std::getenv("LOGPATH");
    if (configPath != nullptr) {
        logPath = std::string(configPath) + "/testing.log"; // Use the directory from LOGPATH
    }

    logFile.open(logPath, std::ios::out | std::ios::app);
}

Logger::~Logger() {
    if (logFile.is_open()) {
        logFile.close();
    }
}

Logger& Logger::getInstance() {
    static Logger instance;
    return instance;
}

void Logger::log(const std::string& message, const std::string& location, Severity severity) {
    std::lock_guard<std::mutex> lock(mtx);

    // Get current time
    auto now = std::chrono::system_clock::now();
    auto now_time_t = std::chrono::system_clock::to_time_t(now);
    auto now_localtime = *std::localtime(&now_time_t);

```

```
42      if (logFile.is_open()) {
43          logFile << "[" << std::put_time(&now_localtime, "%Y
                -%m-%d %H:%M:%S") << "] "
44                  << "[" << severityToString(severity) << "] "
45                  << location << ": " << message << std::endl;
46      }
47  }
48
49  std::string Logger::severityToString(Severity severity) {
50      switch (severity) {
51          case Severity::Info:
52              return "INFO";
53          case Severity::Warning:
54              return "WARNING";
55          case Severity::Error:
56              return "ERROR";
57          default:
58              return "UNKNOWN";
59      }
60  }
```